Brandenburgische
Technische Universität
Cottbus - Senftenberg

Faculty of Mathematics, Natural Sciences and Computer Science

Institute of Computer Science

COMPUTER SCIENCE REPORTS

Report 02/14

July 2014

STOCHASTIC SIMULATION EFFICIENCY

Chiru Swapnil

Computer Science Reports
Brandenburg University of Technology Cottbus - Senftenberg
ISSN: 1437-7969

Send requests to: BTU Cottbus - Senftenberg
Institut für Informatik
Postfach 10 13 44
D-03013 Cottbus

Chiru Swapnil,
http://dssz.informatik.tu-cottbus.de

Stochastic Simulation Efficiency

Computer Science Reports
02/14
July 2014

Brandenburg University of Technology Cottbus - Senftenberg

Faculty of Mathematics, Natural Sciences and Computer Science

Institute of Computer Science

Computer Science Reports
Brandenburg University of Technology Cottbus - Senftenberg
Institute of Computer Science

Head of Institute:
Prof. Dr. Petra Hofstedt hofstedt@tu-cottbus.de
BTU Cottbus - Senftenberg
Institut für Informatik
Postfach 10 13 44
D-03013 Cottbus

Research Groups:	Headed by:
Computer Engineering	Prof. Dr. H. Th. Vierhaus
Computer Network and Communication Systems	Prof. Dr. H. König
Data Structures and Software Dependability	Prof. Dr. M. Heiner
Database and Information Systems	Prof. Dr. I. Schmitt
Programming Languages and Compiler Construction	Prof. Dr. P. Hofstedt
Software and Systems Engineering	Prof. Dr. C. Lewerentz
Theoretical Computer Science	Prof. Dr. K. Meer
Graphics Systems	Prof. Dr. D. Cunningham
Systems	Prof. Dr. R. Kraemer
Distributed Systems and Operating Systems	Prof. Dr. J. Nolte
Internet-Technology	Prof. Dr. G. Wagner

CR Subject Classification (1998): I.6.3, I.6.8, G.4, G.3, D.2.2, D.2.8, D.4.8

Printing and Binding: BTU Cottbus - Senftenberg

ISSN: 1437-7969

Stochastic Simulation Efficiency

Internship Report

May 15, 2014 to July 15, 2014

Author:

Chiru Swapnil

Department of Computer Engineering,
Sardar Vallabhbhai National Institute of Technology, Surat, India

Supervisors:

Prof. Dr.-Ing Monika Heiner
Dipl. Inf. Christian Rohr

Department of Computer Science,
Brandenburg Technical University, Cottbus, Germany

Thursday 14th August, 2014

Abstract

The work described in this report can be broadly divided into two sections.

The first section considers two export features. We describe how the export for stochastic Petri nets to SBML level 1 has been added to the Petri net modelling and simulation tool Snoopy. This task was accomplished by making appropriate changes to the existing export code to generate SBML level 2. Also we demonstrate in detail, how the direct export for coloured Petri nets to both levels (i.e. 1 and 2) of SBML was realised.

The next section summarises the performed comparison of different stochastic simulation tools for biochemical reaction networks. We first compare BioNetGen and SSC with each other by performing simulations on non-coloured Petri nets. Then, we compare the remaining four tools, i.e. Cain, Marcie, Snoopy and Stochkit with each other by performing simulation on coloured Petri nets.

This work builds on results by Aman Sinha [19].

Keywords: Petri Nets, coloured Petri nets, stochastic Petri nets, SBML, stochastic simulation, biochemical reaction networks, export, MathML.

Contents

1 Introduction **2**

2 Exports **3**
2.1 Export for Stochastic Petri Nets to SBML Level 1 3
2.1.1 Text-string math notation v/s MathML subset 3
2.1.2 Pre-defined maths function v/s user-defined function . 7
2.1.3 Reserved v/s non-reserved namespace for annotations . 8
2.1.4 Non-controlled v/s RDF-based-controlled annotation scheme 10
2.1.5 No discrete v/s discrete events 11
2.1.6 Code changes 11
2.2 Export for Coloured Petri Nets 12

3 Tools **14**
3.1 BioNetGen 14
3.2 SSC 17
3.3 CAIN 19
3.4 MARCIE 22
3.5 SNOOPY 24
3.6 StochKit 27

4 The Benchmark Suite **29**
4.1 ANGIOGENESIS 30
4.2 ERK 32
4.3 LEVCHENKO 34
4.4 GRADIENT 36
4.5 REPRESSILATOR 38

5 Performance Comparision **40**
5.1 Results 1 - BioNetGen vs SSC for the uncoloured benchmarks 41
5.1.1 Benchmark Angiogenesis 43
5.1.2 Benchmark Erk 46
5.1.3 Benchmark Levchenko 49
5.1.4 Conclusion 52
5.2 Results 2 - Cain, Marcie, Snoopy and Stochkit for the coloured benchmarks 53
5.2.1 Benchmark Gradient 56
5.2.2 Benchmark Repressilator 64
5.2.3 Conclusion 78

6 Summary **79**
6.1 Achievements 79
6.2 Open Problems 79

References **80**

Appendices **82**

A Accuracy **82**
A.1 Correctness of exports. 82

B How to reproduce the results ? **83**
B.1 BioNetGen 83
B.2 SSC 85
B.3 Cain 88
B.4 Snoopy 89
B.5 Marcie and StochKit 90

C Everything you need to know about Plots. **91**
C.1 Runtime, Memory Consumption and Disk Consumption Plots 91

List of Figures

1 CAIN Screenshot 20
2 Snoopy Screenshot 25
3 Petri net representation of the ANGIOGENESIS model. . . . 31
4 Petri net representation of the ERK model. 33
5 Petri net representation of the LEVCHENKO model. 35
6 Petri net representation of the GRADIENT model, generated out of Figure 7 with $D = 5$ by help of Snoopy. 36
7 A colored Petri net model for the Gradient. 37
8 The repressilator Petri net for three genes in a regulatory cycle. 38
9 A colored Petri net model for the repressilator 39
10 ANGIOGENESIS, Simulation time comparison. 45
11 ANGIOGENESIS, Disk Consumption comparison. 45
12 ERK, Simulation time comparison. 48
13 ERK, Disk Consumption comparison. 48
14 Levchenko, Simulation time comparison. 51
15 LEVCHENKO, Disk Consumption comparison. 51
16 Gradient, Simulation time comparison for Thread=1. 58
17 Gradient, Simulation time comparison for Thread=4. 59
18 Gradient, Simulation time comparison for Thread=8. 59
19 Gradient, Simulation time comparison for Thread=16. 60
20 Gradient, Peak Memory comparison for Thread=1. 61
21 Gradient, Peak Memory comparison for Thread=4. 62
22 Gradient, Peak Memory comparison for Thread=8. 62
23 Gradient, Peak Memory comparison for Thread=16. 63
24 REPRESSILATOR, Simulation time comparison for Thread=1. 72
25 REPRESSILATOR, Simulation time comparison for Thread=4. 73
26 REPRESSILATOR, Simulation time comparison for Thread=8. 73
27 REPRESSILATOR, Simulation time comparison for Thread=16. 74
28 Repressilator, Peak Memory comparison for Thread=1. 75
29 Repressilator, Peak Memory comparison for Thread=4. 76
30 Repressilator, Peak Memory comparison for Thread=8. 76
31 Repressilator, Peak Memory comparison for Thread=16. 77

Task

In this era, SBML is the de facto standard for representing computational models in systems biology. It therefore becomes increasingly important for one's software or tool to support export to and import from SBML. Otherwise, that software or tool is deemed to be outdated.

Hence my first task was to add the export feature to Snoopy so that it can export non-coloured as well as coloured Petri nets to SBML level 1 and 2. (We already had export for non-coloured Petri nets to SBML level 2; so this was not performed).

Also, stochastic modelling and simulation is gaining increasing attention in systems biology. There are many software tools available which are used for stochastic simulation in the domain of biochemical reaction networks. Each tool was developed with a specific objective in mind, and most tools announce themselves to be highly efficient.

So my next task was to compare performance results of some of these tools (BioNetGen and SSC). Also, I was asked to extend the work, done by Aman Sinha [19]. Just for the record, he was specifically working on the comparison of stochastic simulation tools.

1 Introduction

So here we are. Let's see from where to start. Okay, I will give you a brief background of this project, then I will tell you about its motivation and then the outline. Sounds good. So sit back tight, because this report will surely surprise you with amazing results.

Background : There are numerous stochastic simulation tools developed for performing simulation on biochemical reaction networks. A large variety of modeling techniques are used to model the biochemical reaction networks such as Boolean networks, Differential equations (ordinary or partial), Petri nets, etc. Petri nets are found to be a particularly suitable representation of these biochemical reaction networks. For basic understanding about Petri net models please refer [14].

And then, we have SBML. The Systems Biology Markup Language (SBML) is a representation format, based on XML, for communicating and storing computational models of biological processes. It is a free and open standard with widespread software support and a community of users and developers. SBML can represent many different classes of biological phenomena, including metabolic networks, cell signaling pathways, regulatory networks, infectious diseases, and many others.

Motivation : As said earlier, there are a number of stochastic simulation tools available on the market. And each one claims to be at the top of the tree. So it becomes important for us to compare these tools.

Outline : Having realised the exports, we make a performance comparison of tools based on certain comparison criteria (which is explained later in this report). We perform simulation on some benchmark models. Simulations are carried out on each of the selected tool. The results obtained by each tool are compared. The latter portion of this report is an extension of work which was carried out by Aman Sinha [19].

Note : In this report you will find the use of blue colour. Like the one which I just used. This colour is for my readers, if anything is so important that I don't want you to miss it, I will paint that text in blue.

2 Exports

Two types of SBML [11] exports have been considered: export of (plain, i.e., non-coloured) Petri nets, and the export of coloured Petri nets, which require an unfolding first. Both exports will be discussed in the following two sections.

2.1 Export for Stochastic Petri Nets to SBML Level 1

We already have export from stochastic Petri nets to SBML level 2. So before we start, lets understand what are the differences between SBML level 1 and SBML level 2

This will help us not only in making the necessary changes to SBML level 2 codes, but with this we will also save huge amount of time in performing export to SBML level 1.

SBML Level 1	SBML Level 2
Text-string math notation	MathML subset
predefined math functions	user-defined functions
reserved namespaces for annotations	no reserved namespaces for annotations
no controlled annotation scheme	RDF-based controlled annotation scheme
no discrete events	discrete events

Table 1: Differences between SBML levels 1 and 2

If you want to learn more, please visit :- `http://sbml.org/Documents/FAQ#What_are_the_differences_between_Levels_1_and_2.3F`.

In the following we will discuss each point in Table 1 in detail. So let's start with our first point of differences.

2.1.1 Text-string math notation v/s MathML subset

Formulas are used in the definitions of kinetic laws and in rules. When a species name occurs in a formula, it represents the concentration (i.e., substance/volume) of the species. When a compartment name occurs in a formula, it represents the volume of the compartment. The formula strings may contain operators, function calls, symbols, and white space characters. The allowed white space characters are tab and space.

Formulas in SBML Level 1 are expressed in text string form. Mathematical formulas in SBML Level 1 are not expressed using MathML (Maths Markup Language), an XML-based mathematical formula language as it would require simulation software to use fairly complex parsers to read and write the resulting SBML.

Level 1: Example in Rule Use

```
<model>
    . . .
    <listOfRules>
        <parameterRule name="k" formula="k2/k3" >
        <speciesConcentrationRule species="s2" formula="k*z/(1+k)" >
        <compartmentVolumeRule compartment="A" formula="0.10*k4" >
    < /listOfRules>
    . . .
< /model>
```

Level 1: Example in Kinetic Law Use

```
<model>
    . . .
    <listOfReaction>
        <reaction name="J1">
            <listOfReactants >
                <speciesReference species="X">
            < /listOfReactants >
            <listOfProducts >
                <speciesReference species="Y">
            < /listOfProducts >
            <kineticLaw formula="k1*X0"  >
        </reaction>
    </listOfReaction>
</model>
```

Mathematical expressions in SBML Level 2 are represented using MathML 2.0. The XML namespace URI for all MathML elements is

http://www.w3.org/1998/Math/MathML. Note that MathML elements for representing partial differential calculus are not included. The Table 2 describes the subset or the elements used in MathML.

token	cn, ci, csymbol, sep
general	apply, piecewise, piece, otherwise, lambda
relational operators	eq, neq, gt, lt, geq, leq
arithmetic operators	plus, minus, times, divide, power, root, abs, exp, ln, log, floor
logical operators	and, or, xor, not
qualifiers	degree, bvar, logbase
trigonometric operators	sin, cos, tan, sec, csc, cot, sinh, cosh, tanh, sech etc...
constants	true, false, notanumber, pi, infinity, exponentiale
annotation	semantics, annotation, annotation-xml

Table 2: Subset for MathML

Numbers and cn elements

Within MathML expressions contained in SBML (and only within such MathML expressions), numbers in scientific notation must take the form

<cn type = "e-notation"> 2 </sep> -5 </cn>,

and everywhere else they must take the form

$$2e - 5$$

Literal numbers appearing within MathML content in SBML have no declared units.

Boolean values

In XML Schema, the value space of type boolean includes true, false, 1, and 0, whereas in MathML, only true and false count as boolean values.

Csymbol elements

MathML csymbol element is used to denote certain built-in mathematical entities without introducing reserved names into the component identifier namespace such as simulation time and delay function.

The XML fragment below encodes the formula x + t, where t stands for time.

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
            <plus/>
            <ci>x </ci>
            <csymbol encoding="text"
            definitionURL="http://www.sbml.org/sbml/symbols/time">
```

```
            t
            </csymbol>
        </apply>
</math>
```

And, the following XML fragment encodes the equation k + delay(x, 0.1).

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
            <plus/>
            <ci>k </ci>
            <apply>
                <csymbol encoding="text"
                definitionURL="http://www.sbml.org/sbml/symbols/delay">
                delay
                </csymbol>
                <ci>x </ci>
                <cn>0.1 </cn>
            </apply>
        </apply>
</math>
```

Level 2: Example in Rule Use

$$s1 = \frac{T}{1 + k}$$

```
<model>
      . . .
      <listOfRules>
          <assignmentRule variable="s1" >
              < math xmlns="http://www.w3.org/1998/Math/MathML">
                  <apply>
                      <divide/>
                      <ci>T </ci>
                      <apply>
                          <plus/>
                          <cn>1 </cn>
                          <ci>k </ci>
                      </apply>
                  </apply>
              </math>
```

```
        </assignmentRule >
    < /listOfRules>
    . . .
< /model>
```

And, similarily we have to use MathML for kinetic laws.

2.1.2 Pre-defined maths function v/s user-defined function

The basic mathematical functions that are defined in SBML Level 1 at this time are given in Table 3.

Function definitions in SBML level 2 (also informally known as *user-defined functions*) are derived from SBase and contain a math element called Lambda.

Function Definition consists of id and name attribute. The id and name attributes have types SId and String respectively.

The math element is a container for MathML content that defines the function. The content of this element can only be a MathML lambda element or a MathML semantics element containing a lambda element. The lambda element must begin with zero or more bvar elements, followed by any other of the elements in the MathML subset, except lambda (i.e., a lambda element cannot contain another lambda element). This is the only place in SBML where a lambda element can be used.

The number of arguments is equal to the number of bvar elements inside the lambda element of the function definition.

An example showing the definition of cube power in Level 2.

```
<model>
    . . .
    <listOfFunctionDefinition>
        < function Definition id = pow3>
                < math xmlns="http://www.w3.org/1998/Math/MathML">
                    < lambda>
                        <bvar><ci>x </ci> </bvar>
                        <apply>
                            <power/>
                            <ci> x </ci>
                            <ci> 3 </ci>
                        </apply>
```

```
                    < lambda>
                </math>
            </functionDefinition>
        </listOfFunctionDefinition>
        . . .
        <listOfReactions>
            < reaction id="reaction 1">
                . . .
                < kineticLaw>
                    < math xmlns="http://www.w3.org/1998/Math/MathML">
                        <apply>
                            <ci> pow3 </ci>
                            <ci> S1 </ci>
                        </apply>
                    </math>
                < /kineticLaw>
                . . .
            </reacton>
        < /listOfReaction >
        . . .
< /model>
```

2.1.3 Reserved v/s non-reserved namespace for annotations

Annotation element is a container for optional software-generated content not meant to be shown to humans. Every object derived from SBase can have its own value for annotation.The use of XML Namespaces permits multiple applications to place annotations on XML elements of a model without risking interference or element name collisions.

The application developers should choose a URI (Universal Resource Identifier; Harold and Means 2001; W3C 2000a) reference that uniquely identifies the vocabulary that the application will use for such annotations, and a prefix string to be used in the annotations.

Although XML Namespace names (http://www.sbml.org/2001/ns/basis/ for example) must be URIs references, an XML Namespace name is not required to be directly usable in the sense of identifying an actual, retrieval document or resource on the Internet [2]. The name is simply intended to enable unique identification of constructs, and using URIs is a common and simple way of creating a unique name string.

Name	Meaning or Function
abs	absolute value of x
acos	arc cosine of x in radians
asin	arc sine of x in radians
atan	arc tangent of x in radians
ceil	smallest number not less than x whose value is an exact integer
cos	cosine of x
exp	e^x where e is the base of the natural logarithm
floor	the largest number not greater than x whose value is an exact integer
log	natural logarithm of x
log10	base 10 logarithm of x
pow	x^y
sqr	x^2
sqrt	$\sqrt{x}$
sin	sine of x
tan	tangent of x

Table 3: Predefined functions in SBML level 1

However, SBML Level 2 Version 4 places the following restrictions on annotations:

- Within a given SBML annotation element, there can only be one top-level element using a given namespace. An annotation element can contain multiple top-level elements but each must be in a different namespace.
- No top-level element in an annotation may use an SBML XML namespace, either explicitly by refer- encing one of the SBML XML namespace URIs or implicitly by failing to specify any namespace on the annotation. As of SBML Level 2 Version 4, the defined SBML namespaces are the following URIs:
  - http://www.sbml.org/sbml/level1
  - http://www.sbml.org/sbml/level2
  - http://www.sbml.org/sbml/level2/version2

http://www.sbml.org/2001/ns/basis	http://www.sbml.org/2001/ns/jigcell
http://www.sbml.org/2001/ns/biocharon	http://www.sbml.org/2001/ns/jsim
http://www.sbml.org/2001/ns/bioreactor	http://www.sbml.org/2001/ns/libsbml
http://www.sbml.org/2001/ns/biosketchpad	http://www.sbml.org/2001/ns/mathsbml
http://www.sbml.org/2001/ns/biospice	http://www.sbml.org/2001/ns/mcell
http://www.sbml.org/2001/ns/cellerator	http://www.sbml.org/2001/ns/netbuilder
http://www.sbml.org/2001/ns/copasi	http://www.sbml.org/2001/ns/pathdb
http://www.sbml.org/2001/ns/cytoscape	http://www.sbml.org/2001/ns/promot
http://www.sbml.org/2001/ns/dbsolve	http://www.sbml.org/2001/ns/sbedit
http://www.sbml.org/2001/ns/ecell	http://www.sbml.org/2001/ns/sigpath
http://www.sbml.org/2001/ns/gepasi	http://www.sbml.org/2001/ns/stochsim
http://www.sbml.org/2001/ns/isys	http://www.sbml.org/2001/ns/vcell
http://www.sbml.org/2001/ns/jarnac	http://www.sbml.org/2001/ns/jdesigner

Table 4: Reserved XML Namespace names in SBML Level 1 Version 2.

  - http://www.sbml.org/sbml/level2/version3
  - http://www.sbml.org/sbml/level2/version4

- The ordering of top-level elements within a given annotation element is not significant.

2.1.4 Non-controlled v/s RDF-based-controlled annotation scheme

SBML Level 1 does not provide this feature, whereas SBML Level 2 provides us with the RDF feature.

This format described in Level 2 is intended to be the form of one of the top-level elements that could reside in an annotation element attached to an SBML object derived from Sbase. The element is named rdf:RDF. The SBML structures described elsewhere in this document do not have any biochemical or biological semantics. The format described in this SBML Level 2 provides a scheme for linking SBML structures to external resources so that those structures can have such semantics.

2.1.5 No discrete v/s discrete events

There are no discrete events in SBML Level 1, whereas SBML Level 2 has discrete events.

2.1.6 Code changes

So after looking at the differences, we are ready to perform the export to SBML level 1. This has been done by applying the required changes to the already existing export for SBML level 2.

- What we actually want is, that we have a GUI dialog box which asks the user to choose between the SBML levels i.e. 1 and 2. And based on his/her choice the export to either of the level is performed.

  To incorporate this feature we added two fuctions *AddToDialog()* and *OnDialogOk()*, respectively. The first function generates radio buttons one for level 1 and another for level 2. And based upon, which radio button is checked, the export is performed.

  OnDialogOk() function assigns the value of 1 or 2 to the variable *level*.

- In the *DoWrite()* function we made two major changes.

  First, based on the value of variable *level* we added appropiate level and version value to the XML file.

  And second, *Unit Definition* component was added. This component in the XML file is a convenient way of defining new units. And under this component, a unit named *substance* was created.

  Note: However this component is not necessary as such, I mean both levels will work fine without this, but Dizzy (a tool, which supports only SBML Level 1) requires this component in the XML file. Therefore, this component was added to both SBML Levels.

- After going through the differences between the two levels, we know that SBML Level 1 does not support MathML. So in *WriteTransition()* function changes were made in the kineticLaw code. A code snippet below gives an idea of the changes made.

```
if(level==1)
{
  KineticLaw*  l_pcKineticLaw =
                              l_pcReaction->createKineticLaw();
  l_pcKineticLaw->setFormula(l_sEquation.utf8_str().data());
```

```
}
else   //for level 2 by default
{
  KineticLaw* l_pcKineticLaw =
                              l_pcReaction->createKineticLaw();
  l_pcKineticLaw->setMath(SBML_parseFormula(
                              l_sEquation.utf8_str().data()));
}
```

For level 1, directly the Kinetic formula in text string format is used, while in Level 2, the kinetic formula is parsed and set to MathML form.

- Then in *ValidateSBML()* function, based on the value of variable *level* we check the SBML Document compatibility. If no errors are found, the export to SBML is made.

- No changes were made in *WritePlaces()*, *WriteConstants()*, *WriteParameters()* and *AcceptsDoc()* functions.

2.2 Export for Coloured Petri Nets

This export has been done for both SBML levels. We will discuss all the functions in detail, which were written in order to perform the export. So let's start.

- *AddToDialog()* function provides us with the radio buttons so that we can select appropriate Level.

- *OnDialog()* function assigns the appropriate value to variable *level*, depending upon which radio button is selected.

- *AcceptDoc()* function checks if the right graphs are given and will only accept graphs which are coloured Petri nets, coloured extended Petri nets, coloured stochastic Petri nets, coloured continuous petri nets, and coloured hybrid Petri nets.

- *DoWrite()* function first performs the unfolding of coloured Petri nets. Then, based on the value of variable *level* it writes down the appropriate level and version in the XML file and then the components, namely *UnitDefinition* and *Compartment* is added in the XML file.

- *WriteConstants()* function are only called for coloured stochastic, coloured hybrid and coloured continuous Petri nets. This function iterates through

all the constants in the graph and writes them in the XML file. These constants are written in the *Parameter* component of the XML file and its attributes, namely Constant, Id and Value, are set accordingly.

- *WritePlaces()* and *WritePlaceClass()* function unfold all the places in the graph, and then they add the *Species* component in the XML file with its attribute namely, Compartment, HasOnlySubstance, Id and InitialAmount which are set accordingly.

- *WriteTransition()* and *WriteTransitionClass()* function unfolds all the transitions in the graph and then, it adds the *Reaction* component for each transition in the XML file. Based on the Level of export, *KineticLaw* is added.

  Note- This Program will write Kinetic Law in the XML file in the form of Mass-Action. So use the explicit version while doing the export.

- *WriteTransitionClass()* function calls the *WriteArc()* function which then adds *Reactant* and *Product* sub-component under *Reaction* section. *WriteArc()* function also adds *Modifier* component in case it encounters special arcs (like inhibitor, equal arcs etc..).

- *WriteParameters()* function iterates through all the parameters in the graph and create *Parameter* component in the XML file.

- Then in *ValidateSBML()* function, based on the value of variable *level* we check the SBML Document compatibility. If no errors are found, Export to SBML is made.

3 Tools

In this chapter we summarise the tools which are used for simulating the benchmark models and which are used in the actual performance comparison. The following tools (namely : *Cain, Marcie, Snoopy, Stochkit*) had already been used in [19]. To be self-contained, we repeat their description in this report as well. The new tools additionally considered here are: *BioNetGen and SSC*. We start with the new tools first in lexical order and then we will go through the remaining tools (in lexical order).

3.1 BioNetGen

- The BioNetGen software package was initially developed by the Cell Signaling Team at Los Alamos National Laboratory. The current development team is based in the Department of Computational and Systems Biology at the University of Pittsburgh School of Medicine, with contributions from collaborators at the Theoretical Division and Center for Nonlinear Studies at Los Alamos National Laboratory, the Departments of Biology and Computer Science at the University of New Mexico, the Center for Cell Analysis and Modeling at the University of Connecticut Health Center, and the Department of Biological Chemistry at the Johns Hopkins University School of Medicine. This tool can be downloaded from

  `http://bionetgen.org/index.php/BioNetGen_Distributions` .
  And in order to perform the simulation the following link was used

  `http://bionetgen.org/index.php/BNGManual:Simulating_a_Network`

- Modelling Paradigm:

  1. Simulate reaction networks as a set of ODEs.
  2. Simulate reaction network using Gillespies "stochastic simulation algorithm".
  3. Simulate reaction network using the partitioned-leaping algorithm (tau-leaping variant).
  4. Simulate rule-based model using network-free stochastic simulator NFsim

- Model class: This discussion is beyond the scope of this report.

- Data exchange formats: Imports and Exports

Import file formats - .bngl, .net

Export file formats -

- Write rule-based model in BNGL format.
- Write rule-based model in BNG-XML format (read by NFsim).
- Write reaction network in NET format.
- Write reaction network in SBML format.
- Write reaction network in MatLab format.
- Write network-specific CVode integrator with Matlab interface.
- Write reaction network in MDL format for CellBlender/MCell.

- Tool features and handling: The tool was found to be easy in handling. All the necessary commands which are used while performing simulations and running the BioNetGen tool are given in the link `https://docs.google.com/spreadsheet/ccc?key=0Avcdx-KzjXH4dGhLZWlZZ1VGSmYzb0ZvRG0za3I gid=0`

  `http://bionetgen.org/index.php/Installation_Guide`

  respectively.

  One main feature of this tool, or rather I should say a *major drawback of BioNetGen* is that it can only perform a single simulation run. Unfortunately, there's not a simple argument that you can pass for running multiple simulations. But if we use 'parameter_scan' action, for multiple simulation then each simulation run will generate namely .cdat, .gdat and .net files respectively for each run. Therefore if a simulation is performed for a *million* runs, then it would generate *three million* files which requires a huge amount of disk space.

- Interface: It is a command line tool.

- Evaluation of results: Whenever any simulation (like ssa) is performed, Trajectory data are written into two multicolumn output files for each simulation: a .gdat file that reports the value of each defined observable at each sample time and a .cdat file that reports the population level of every species in the network at each sample time. Both data file types are in ASCII format, so they can be viewed in a text editor or imported into any number of different plotting and data analysis programs. The BioNetGen distribution includes the PhiBPlot plotting utility, which is a Java program that can be run by double-clicking on the file PhiBPlot.jar in the PhiBPlot subdirectory of the distribution

or by typing "java -jar path/PhiBPlot.jar [datafile]" on the command line.

- Implementation Language: BioNetGen-2.x.x is written in the Perl language. The simulation back end, run_network, is written in C++. The PhiBPlot plotting utility, which is included in the PhiBPlot subdirectory of the distribution, is written in Java.
- Platforms :Running BioNetGen requires-
  - Perl version 5.8 or above. This is usually installed on Mac OS X and Linux machines and under Windows if you are running Cygwin.
  - Mac OS/X, Linux, Windows, or a platform with appropriate tools for compiling the simulation backend.
- License: This is available free of cost for non-commercial use.
- Tool version: 2.2.5
- Ease of installation: The installation for this tool was found to be easy. And the steps for installing the tool can be found on `http://bionetgen.org/index.php/Installation_Guide`

3.2 SSC

- This tool was developed at MIT and can be downloaded from

  `http://web.mit.edu/irc/ssc/`

  Inorder to perform simulation following video was found helpful which is given in the SSC website

  `http://web.mit.edu/irc/ssc/downloads/ssc-viewer-screencast.mov`.
  However the video as well as Installation manual say that they support Mac OS X, but unfortunately they no longer provide a Mac OS X release.

- Modelling Paradigm: As the name suggest the Stochastic Simulation Compiler (SSC) is a tool for creating exact stochastic simulations of biochemical reaction networks.SSC compiles the model into fast simulators. Part of the speedup comes from algorithmic improvements to the original Gillespie algorithm, while the rest comes from directly generating efficient native code.

- Model class: This discussion is beyond the scope of this report.

- Data exchange formats:

  1. Imports: .rxn (reaction file) and .cfg (configuration file)
  2. Exports: .trj (trajectory file), matlab (.txt format but for this ssc-trj-reader-0.01.jar is required which has to be downloaded seperately (click on SSC trajectory reader for downloading on te given page `http://web.mit.edu/irc/ssc/`))

- Tool features and handling: The tool was not easy to handle, but it was not difficult also. It was somewhere in between easy and difficult. SSC also has the same *drawback* as BioNetGen i.e. it can only perform a single simulation run and if we write a script for multiple simulation run, then each run would generate its own file, and which requires a large amount of disk space.

- Interface: It is a command-line tool.

- Evaluation of results: The .trj files can be directly plotted using SSC 3D Viewer(this can be downloaded by clicking start direcly link under the SSC 3D viewer section on ths page `http://web.mit.edu/irc/ssc/`)

  Also we have ssc-trj-reader-0.01 (a jar file) which allows converting SSC-generated .trj files to Matlab-readable format.

- Implementation Language: No information was given on the website.
- Platforms: It supports only Linux. However, the installation manual say that they support Mac OS X, but unfortunately they no longer provide a Mac OS X release.
- Hardware architecture: 64 bit version was downloaded and installed.
- License: Copyright 2008 MIT
- Tool version: The version downloaded was ssc-0.6-Linux-x86_64.tar.bz2. It was downloaded from `http://web.mit.edu/irc/ssc/`
- Ease of installation: The installation of SSC was easy. However in order to run SSC 3D Viewer, one has to change the security of java. If the java security is high it won't run on your machine.

3.3 CAIN

- This tool was developed at California Institute of Technology, Pasadena, California, United States. In order to perform simulation the documentation of CAIN was referred which can be found on `http://cain.sourceforge.net/`. The tool is available for download on `http://cain.sourceforge.net/`.

- Modelling Paradigm: It supports stochastic, deterministic as well as hybrid models. Its simulation method include

  1. Discrete Stochastic Simulations
  2. Direct Method
  3. First Reaction Method
  4. Next Reaction Method
  5. Tau-Leaping
  6. SAL Tau-Leaping
  7. Direct Method with Time-Dependent Propensities
  8. Hybrid Direct/Tau-Leaping
  9. ODE Integration

- Model class: This discussion is beyond the scope of this report.

- Data exchange formats: Imports and Exports

  - It stores models, simulation parameters, and simulation results in an XML format. It supports XML import as well as export.
  - In addition, it also supports SBML imports and exports. The level and versions are not explicitly mentioned in the manual.
  - The results generated can be exported in gnu plot files and it also exports the script for gnu plot to plot the result file.
  - There is a csv export of the simulation result which exports result in the csv format.

- The handling of the tool was easy. There are separate panels which make simulation analysis easy. The complete model is described in a single window within their respective panels. E.g. Model Panel, Method Panel, Reaction Panel, Species Panel etc.

- Interface: It is a GUI tool.

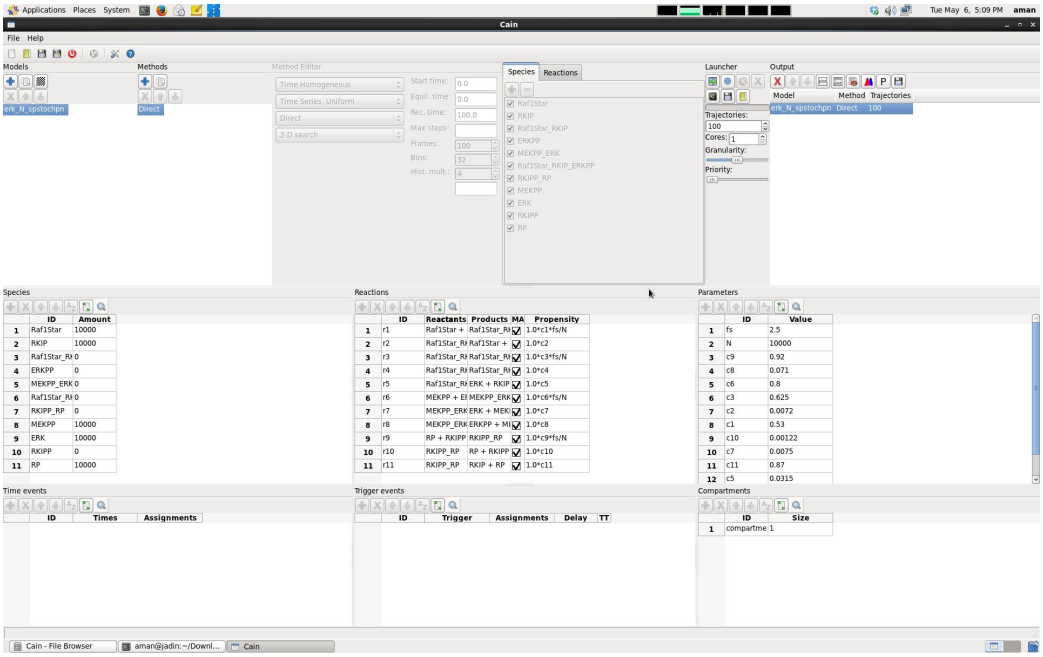

Figure 1: CAIN Screenshot

- Evaluation of results: CAIN can plot its result by plotting Time Series Data, plotting histograms and tables. It does not support model checking.
- Parallel Computing: Yes, implementation principle unknown.
- Implementation Language: The GUI is written in Python and uses the wxPython toolkit. However the solvers are written in C++ and are implemented as command line executables.
- Platforms: It supports all the three platforms namely Linux, Windows and Mac/OS.
- Hardware architecture: 64 bit version was downloaded and installed. The type of architecture is nowhere mentioned exclusively.
- License: Copyright (c) 1999 to the present, California Institute of Technology
- Tool version: The version downloaded was version 1.10. It was downloaded from `http://sourceforge.net/projects/cain/files/cain/` which was made available on sourceforge on 2 July 2012. The website

of cain is not updated, it says the latest release is version 1.9 on 27 September 2011. The tool was downloaded on 02 April 2014.

- Ease of installation: The link for CAIN can be found at the SBML website:
  http://sbml.org/SBML_Software_Guide/SBML_Software_Summary#cat_9
  The above link directs to the CAIN website on
  http://cain.sourceforge.net/
  The download button on the last link will re-direct to
  http://sourceforge.net/projects/cain/
  from where CAIN can be downloaded. The zip file is downloaded and extracted. The documentation on the cain website was read and steps to install CAIN on REDHAT 6.0/CENTOS 6.0 were followed.
  CAIN requires C++ compiler which was already installed on my system. CAIN requires Python, wxPython, matplotlib, numpy and sympy. The easiest way to install the above mentioned package is to install the Enthought Python Distribution. It includes all the packages which CAIN requires. The Enthought Canopy can be downloaded from
  https://www.enthought.com/downloads/
  The installation guide for Canopy was also read which can be found on
  http://docs.enthought.com/canopy/quick-start/install_linux.html
  After performing these steps we have sufficient packages installed on the system for CAIN. The CAIN package which was downloaded from
  http://sourceforge.net/projects/cain/
  was unzipped. The installation instruction for CAIN is available at
  http://www.cacr.caltech.edu/~sean/cain/InstallationLinux.htm
  The overall installation was easy.

3.4 MARCIE

- MARCIE stands for (M)odel checking (A)nd (R)eachability analysis done effi(CIE)ntly. MARCIE is a tool for qualitative and quantitative analysis of generalized stochastic petri nets with extended arcs. MARCIE is the successor of IDDMC.

- This tool was developed at Brandenburg Technical University, Cottbus, Germany `http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Marcie`. In order to perform simulation the user manual was referred. For user manual please refer [18]. The tool is available for download on `http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Marcie#downloads`

- Modelling paradigm: It is an analysis tool for stochastic petri nets. The engines available are :

  1. Exact Numerical Engine which includes:
     - Jacobi method
     - Gauss-Seidel method
     - Pseudo-Gauss-Seidel method
     - Immediate Transitions
     - Markovian approximation
     - Computation of probability distributions
  2. Approximate Numerical Engine
  3. Simulative Engine

- Model class: This discussion is beyond the scope of this report.

- Data exchange formats: Imports and Exports

  - Input file formats - .apnn, .andl, .pnml
  - The file can be created using the ANDL- export feature of Snoopy.
  - It writes simulation result in CSV format.

- Tool features and handling: The tool was found to be easy in handling. The tool is a command line tool and the all the necessary commands which are used while performing simulations are mentioned in the user manual. The results can be exported to a .csv file.

- Interface: It is a command line tool. While simulation, it displays the progress of the simulation (i.e. how much of the simulation is complete).The total processing time includes the simulation run time as well as the time for writing the file. We are concerned with the total elapsed time because it is the simulation run time. The simulation runtime is expressed in the format of 0m0sec.

- Evaluation of results: The simulation results can be exported to a .csv file which can be processed by gnuplot in order to plot the graph. It does not support any plotting function. It support model checking.

- Parallel computing: Yes, implementation principle unknown.

- Implementation language: MARCIE is written in C++.

- Platforms: It is supported in Linux and Mac/OS. Hardware architecture: Only 64 bit for Linux was downloaded and installed. The current version is available for MAC OS 10.5/6 , Linux32 and Linux64.

- License: This is available free of cost for non-commercial use.

- Tool version: MARCIE was first released on 23 December 2010. The latest release of MARCIE was on 19 July 2012. The latest release of MARCIE was downloaded and used for performing simulation. The tool was downloaded on 17 April 2014.

- Ease of installation: MARCIE can be downloaded from the link : `http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Marcie#download`
  The downloaded file can be extracted and MARCIE can be run directly by going into the sub-folder.
  However MARCIE requires GLIBC version 2.14 and GLIBCXX 3.4.15 for its execution.
  The installation for this tool was found to be easy.

3.5 SNOOPY

- This tool was developed at Brandenburg Technical University, Cottbus, Germany `http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Snoopy`. In order to perform simulation please refer [14] and [15]. The user manual was also referred. For user manual please refer [10]. For the graph based data structure used in Snoopy and modeling and simulation in Snoopy refer [9]. The tool is available for download on `http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Snoopy#downloads`

- Modelling paradigm: The available simulators are stochastic, deterministic and hybrid. The algorithms available are :

  1. Stochastic Simulators:
     - Gillespie
     - FAU
  2. Continuos Simulators:
     - BDF
     - Rosenbrock-Method of Shampine
     - Rosenbrock-Method GRK4T of Kaps-Rentrop
     - Rosenbrock-Method GRK4A of Kaps-Rentrop
     - Rosenbrock-Method of Van Veldhuizen [gamma = 1/2]
     - Rosenbrock-Method of Van Veldhuizen [D-stable]
     - an L-stable Rosenbrock-Method
  3. Hybrid Simulators:
     - Explicit RK
     - Implicit RK
     - BDF
     - ADAMS

- model class: This is beyond the scope of this report.

- Data exchange formats: Imports and Exports

  - It can import as well as export SBML level 2 version 3.
  - It supports several other imports and exports. For more imports and exports visit the web page `http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Snoopy#imexport`

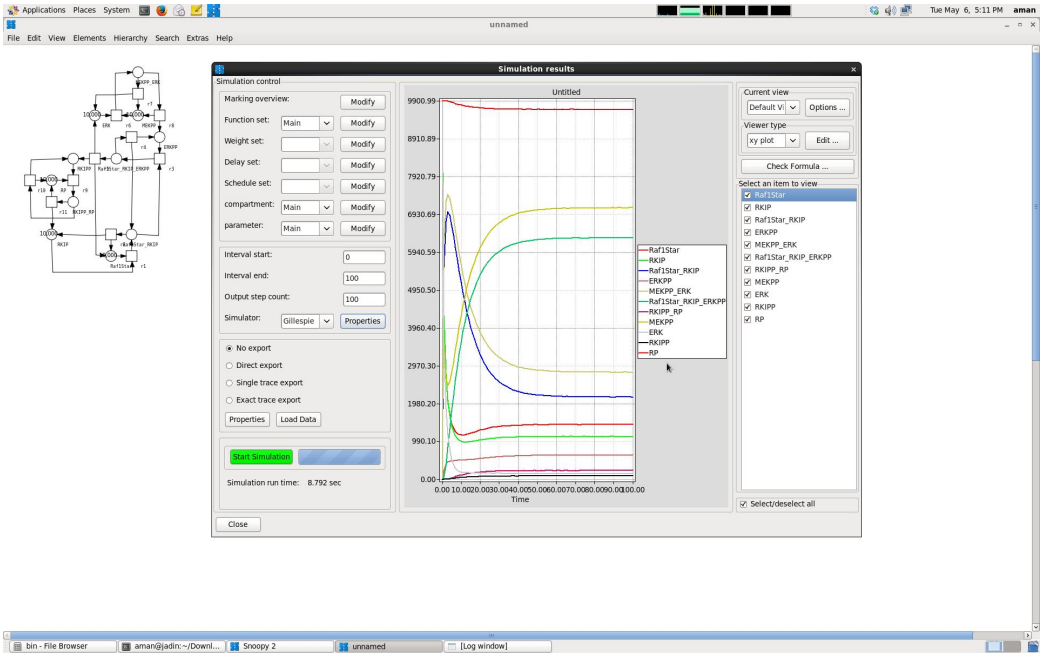

Figure 2: Snoopy Screenshot

- Tool features and handling: The tool was found to be easy in handling. The tool provides a special animation mode where you can play the token game, which helps in better understanding of the model. The simulation window is very easy to handle. The graphs are plotted automatically. The simulation control panel contains the different functions sets, parameters, simulators etc.

- Interface: It is a GUI tool.

- Evaluation of results: The default is the graphical plot which appears on the simulation window. The results can be exported in csv format as well as image can also be exported (e.g. gif, bmp etc). The viewer view panel has three options xy plot, histogram, and tabular. The xy plot shows the graphical lines in the simulation window, the histogram shows the graphical histogram representation in the simulation window and the tabular view shows the result in the tabular format in the simulation window. It does not support model checking.

- Parallel computing: Yes, implementation principle unknown.

- Implementation language: Snoopy is implemented in C++, wxWidgets, Xerces.

- Platforms: It supports all the three platforms namely Linux,Windows and Mac/OS.
- Hardware architecture: Only 64 bit for linux was downloaded and installed. 32 bit version is available for Linux, however the architecture is not explicitly mentioned for Mac and Windows.
- License: This is available free of cost for academic purpose and for non commercial use.
- Tool version: The version downloaded was version 1.13. Snoopy was first released on 9 October 2008. Its latest release was on 01 April 2014. The tool was downloaded on 14 April 2014.
- Ease of installation:

  The installation was found to be easy.

3.6 StochKit

- This tool wass developed at UC Santa Barbara University of California, United States. `http://sourceforge.net/projects/stochkit/`. In order to perform simulation the user manual was referred which is provided with the installation file. The tool is available for download on `http://sourceforge.net/projects/stochkit/`

- Modelling paradigm: StochKit2 provides commandline executables for running stochastic simulations using variants of Gillespies Stochastic Simulation Algorithm and Tauleaping. Improved solvers including efficient implementations of :

  1. SSA Direct Method
  2. Optimized Direct Method
  3. Logarithmic Direct Method
  4. ConstantTime Algorithm
  5. Adaptive Explicit Tauleaping method

- Model class: This discussion is beyond the scope of this report.

- Data exchange formats: Imports and Exports

  - The source file is stored in a .cpp format.
  - Uses a Java Converter to convert the SBML input file to make it compatible with StochKit.
  - The converter accepts the standard version 1 (level 1 and level 2) of SBML and version 2 SBML files.

- Tool features and handling: The tool was found to be easy in handling. The tool is a command line tool and the all the necessary commands which are used while performing simulations are mentioned in the user manual. The results can be exported to a .txt file. It exports means as well as variance of the species in the reaction. It has a special feature of determining the simulation method based on the model that will achieve best performance while simulation.

- Interface: It is a command line tool. It displays the drivers which it uses while performing simulation. The simulation runtime is displayed at the end.

- Evaluation of results: The simulation results can be exported to a .txt file which can be processed by gnuplot in order to plot the graph. It supports plotting function. The plotting tools are available in MATLAB. It does not support model checking.

- Parallel computing: Yes, implementation principle unknown.

- Implementation language: Stochkit is written in C++.

- Platforms: It supports all the three platforms namely Linux, Windows and Mac/OS.

- Hardware architecture: Only 64 bit for Linux was downloaded and installed. There is no explicit mention of the architecture.

- License: StochKit2 (version 2.0.5 and later) is distributed under the BSD 3Clause License (BSD New or BSD Simplified).

- Tool version: The latest release of StochKit is StochKit 2.0.10 on 20 November 2013. The latest release of StochKit was downloaded and used for performing simulation. The tool was downloaded on 02 April 2014.

- Ease of installation: The link for StochKit can be found at the SBML website:
  `http://sbml.org/SBML_Software_Guide/SBML_Software_Summary#cat_9`
  The above link directs to the StochKit website on:
  `http://www.engineering.ucsb.edu/~cse/StochKit/`
  The above link will be directed to sourceforge for the download option:
  `http://sourceforge.net/projects/stochkit/`
  StochKit2 was downloaded and extracted. In the extracted folder there is a StochKit2 manual. The installation steps written in the manual were followed. StochKit was installed successfully.
  However for importing SBML files we need SBML converter. The SBML converter was found in the tools sub-folder. The documentation was read and the steps to install the SBML converter were followed. It needs an additional library libSBML which needs to be installed.After performing the steps written int the documentation file, the SBML converter was installed successfully.
  The installation was found to be difficult.

4 The Benchmark Suite

Model form. The following sections summarise all benchmark examples. The information is structured into:

- *Description.* a brief description of the example including a figure showing the Petri net model, and some references where it has been published.
- *Scaling parameter.* List of parameters and their meaning for model scaling.
- *Model size.* Size of the Petri net model in terms of number of places, transitions and arcs. These numbers have been found by importing the SBML file in Snoopy and viewing the net information.
- *Simulation parameters.* Chosen setting for the simulations, such as interval start time, interval end time, interval steps, value of scalable parameter, number of runs, number of experiments per run and number of threads.

Non-Coloured Petri Nets

- Angiogenesis
- Erk
- Levchenko

Coloured Petri Nets

- Gradient
- Repressilator

To be self-contained, we provide here a description of all benchmarks used. The descriptions for Angiogenesis, Erk and Levchenko have been taken from [19].

4.1 ANGIOGENESIS

Description. Angiogenesis, defined as the formation of new vessels from existing ones, is a topic of great interest in all areas of human biology, particularly to scientists studying vascular development, vascular malformation and cancer biology. Angiogenesis is a complex process involving the activities of many growth factors and relative receptors, which trigger several signaling pathways resulting in different cellular responses. The Petri net was introduced in [16] and refined in [3], see Figure 3.

Scaling parameter

- N – initial number of tokens on places Akt, Enz, Gab1, KdStar, P3k, Pg, Pip2 and Pten

Model size

- number of places: 39
- number of transitions: 64
- number of arcs: 185

Although the model is parameterized, the size of its structure does not depend on parameter values.

Simulation parameters

- interval start time: 0
- interval end time: 100
- interval steps: 100
- value of N: 1, 5, 10, 50
- no of runs: 1, 100, 10,000, 1,000,000

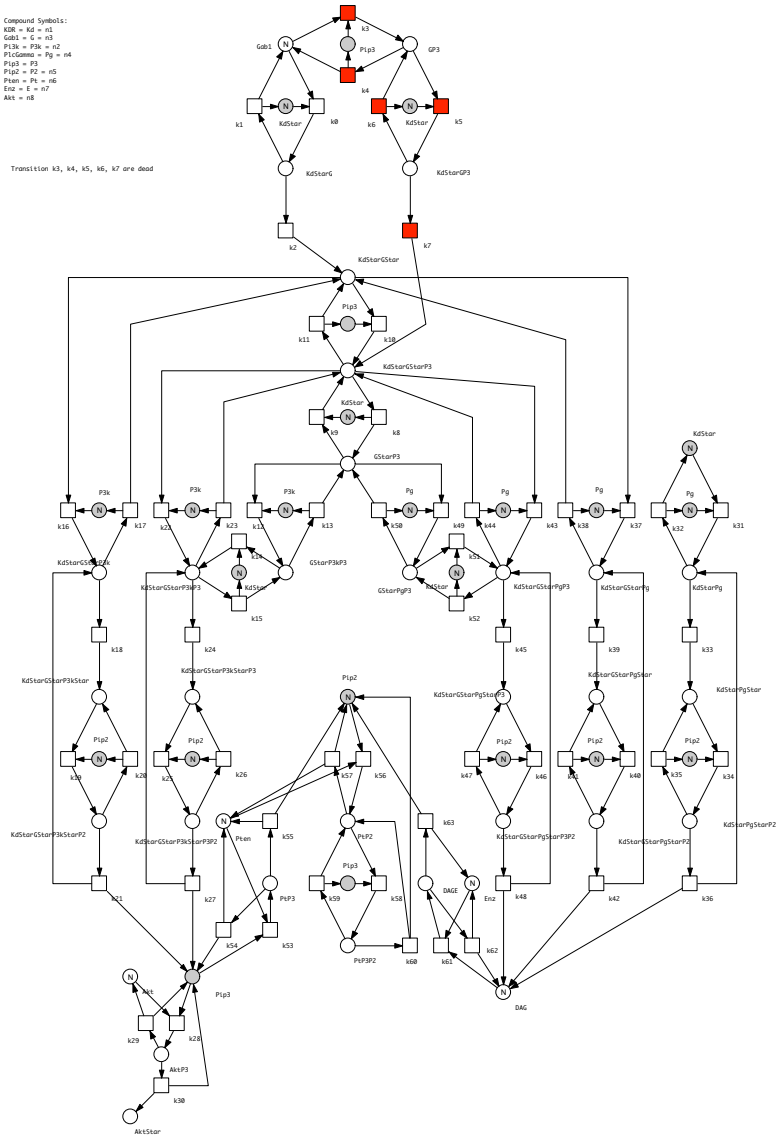

Figure 3: Petri net representation of the ANGIOGENESIS model.

4.2 ERK

Description. The RKIP inhibited ERK pathway was originally published in [12], and discussed as qualitative and continuous Petri nets in [4], and as three related Petri net models comprising the qualitative, stochastic and continuous paradigms in [7], see Figure 4.

Scaling parameter

- N – the initial number of tokens on the places ERK, MEKPP, Raf1Star, RKIP and RP;

Model size

- number of places: 11
- number of transitions: 11
- number of arcs: 34

Although the model is parameterized, the size of its structure does not depend on the parameter values.

Simulation parameters

- interval start time: 0
- interval end time: 100
- interval steps: 100
- value of N: 1, 100, 10,000, 1,000,000
- no of runs: 1, 100, 10,000, 1,000,000

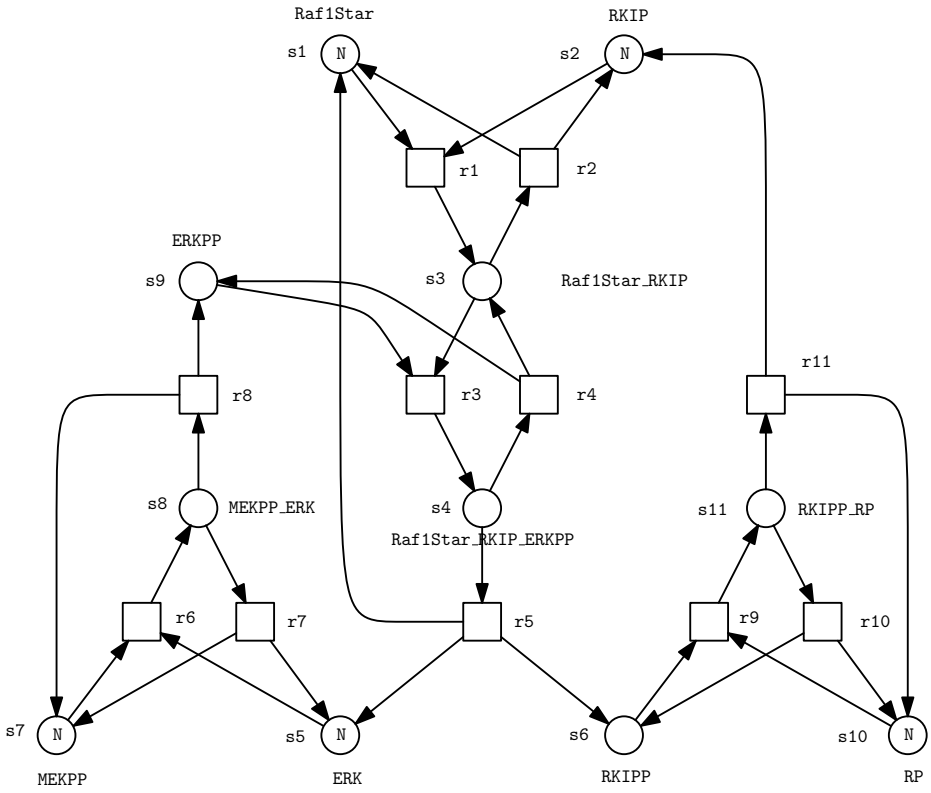

Figure 4: Petri net representation of the ERK model.

4.3 LEVCHENKO

Description. The mitogen-activated protein kinase (MAPK) cascade was published in [13]. This is the core of the ubiquitous ERK/MAPK pathway that can, for example, convey cell division and differentiation signals from the cell membrane to the nucleus. It has been used in [5] and [8] as running example to discuss three related Petri net models comprising the qualitative, stochastic and continuous paradigm, see Figure 5.

Scaling parameter

- N – the multiplier of the initial number of tokens on the places Raf, RasGTP, RafP_Phase1, MEKP_Phase2, ERk, ERKP_Phase3

Model size

- number of places: 22
- number of transitions: 30
- number of arcs: 90

Although the model is parameterised, the size of its structure does not depend on parameter values.

Simulation parameters

- interval start time: 0
- interval end time: 100
- interval steps: 100
- value of N: 1, 10, 100, 1,000
- no of runs: 1, 100, 10,000, 1,000,000

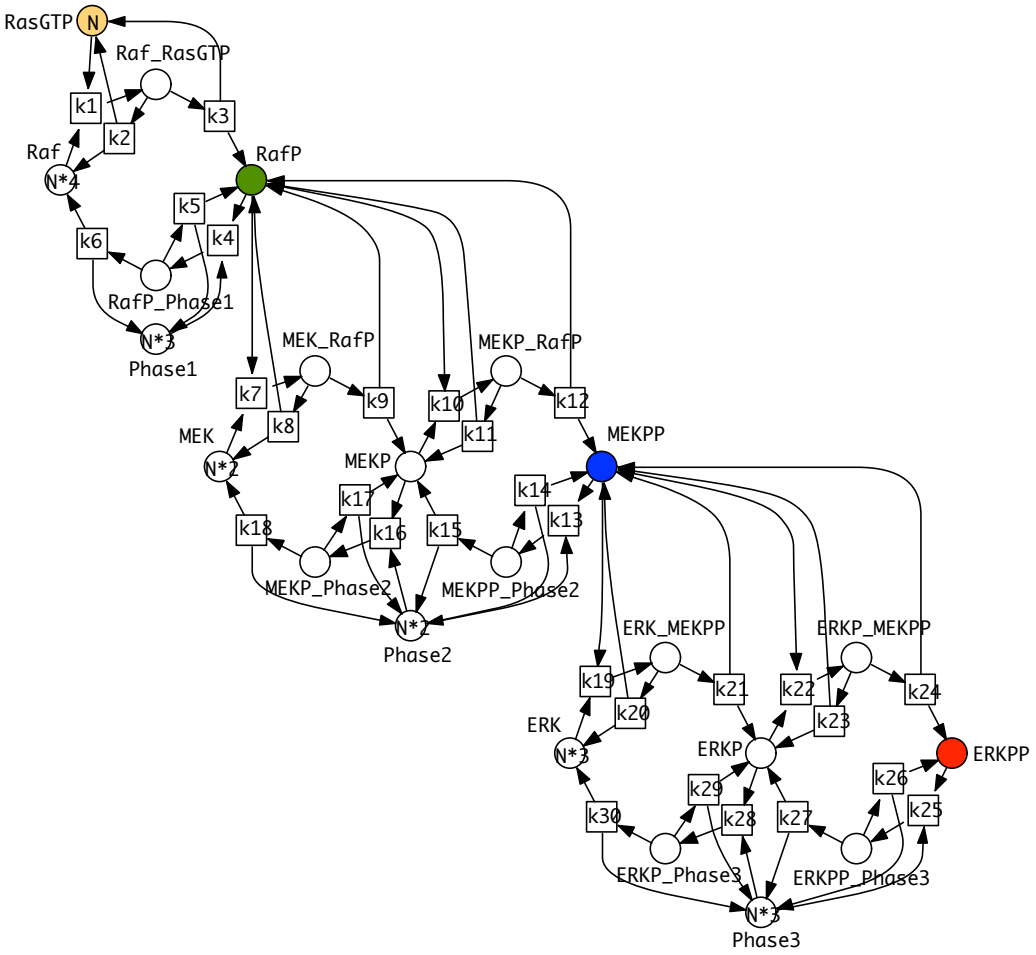

Figure 5: Petri net representation of the LEVCHENKO model.

4.4 GRADIENT

Description. Diffusion in space is a basic process underlying many spatial (bio-) chemical processes, however typically considered either in the stochastic or continuous setting. The Petri net given here comes from [6], where it has been used to illustrate the generic modelling of space by use of coloured Petri nets. We discretise the space by a D X D rectangular grid, D being a model parameter, and deploy the 8-neighbourhood relation with reflecting boundary condition. The process starts with N tokens in the centre position. This model is easily scalable with well-known size of the model growth and its state space; see Figure 6 for an unfolded model version, and see Figure 7 for the scalable coloured gradient model.

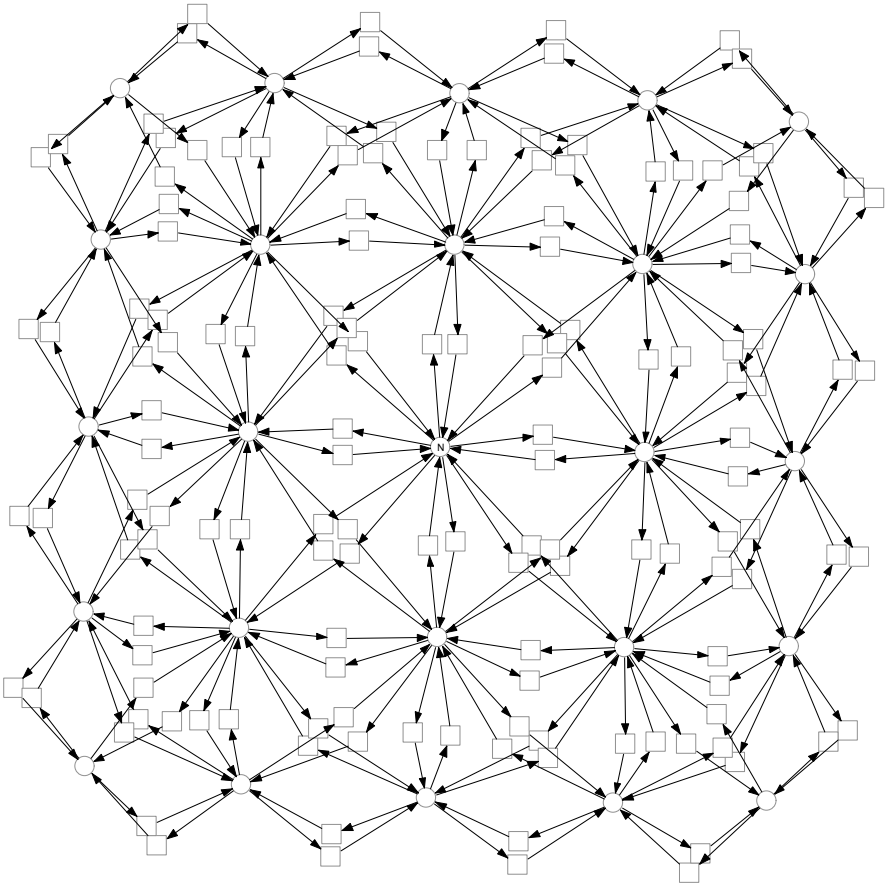

Figure 6: Petri net representation of the GRADIENT model, generated out of Figure 7 with $D = 5$ by help of Snoopy.

Scaling parameter

- D – grid size; i.e. there are $D \times D$ grid positions,

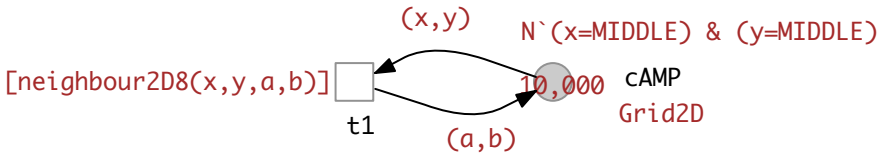

Figure 7: A colored Petri net model for the Gradient.

- N – there are initially $N = 1000 * D$ tokens in the centre position see Figure 6.

Model size

- parameter : D
- number of places : D^2
- number of transitions : $8D^2$ - $12D$ +4
- number of arcs : 2 - $|T|$

The model is parametrized and the size of its structure depends on the parameter D.

Simulation parameters

- interval start time: 0
- interval end time: 100
- interval steps: 100
- value of D: 10, 50
- no of runs: 1, 100, 1000, 10,000
- no of experiments per run: 10 (*No Simulations for Snoopy and Cain due to limited time*)
- no of threads: 1, 4, 8, 16

4.5 REPRESSILATOR

Description. The Petri net given here comes from [1], where it has been used to illustrate a modular and stepwise construction of a Petri net model. When genes repress each other in a circular manner, we obtain a gene regulatory cycle, see Figure 8 which is composed of three gene gates with identical structure. For the coloured repressilator model see Figure 9.

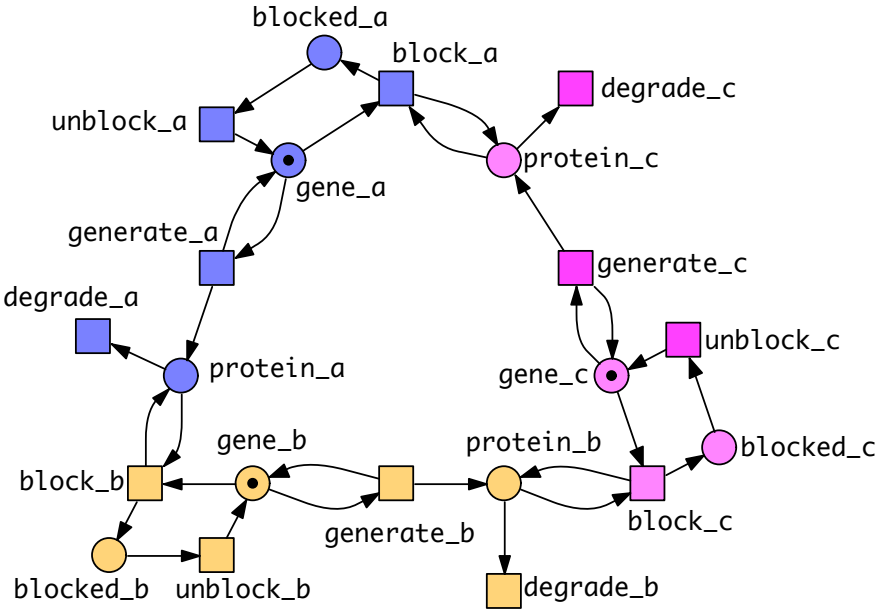

Figure 8: The repressilator Petri net for three genes in a regulatory cycle.

Scaling parameter

- N - initial number of tokens on the coloured place *Gene*, in the Figure 9 we have N = 3 as you can see in the gene place we have three black tokens (dots)

Model size

The model is parameterized and the size of its structure depends on the parameter value, see Table 5.

Simulation parameters

- interval start time: 0
- interval end time: 10000
- interval steps: 100

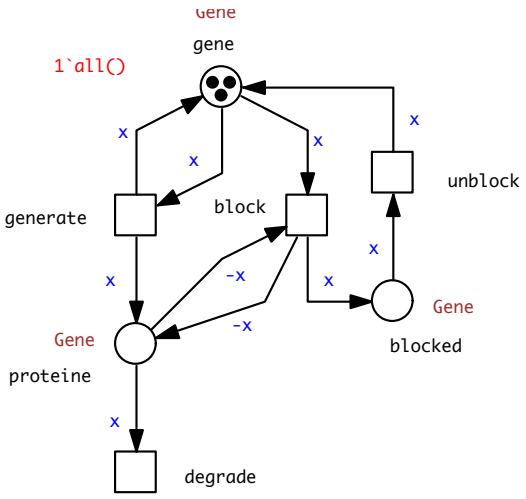

Figure 9: A colored Petri net model for the repressilator

N	Place Number	Transition Number	Standard Arc
3	9	12	30
30	90	120	300
100	300	400	1000
300	900	1200	3000
3000	9000	12000	30000
N	3N	4N	10N

Table 5: Model Size for Repressilator

- value of N: 3, 30, 100, 300, 3000
- no of runs: 1, 100, 1000, 10,000
- no of experiments per run: 10 (*3 for Cain due to limited time*)
- no of threads: 1, 4, 8, 16

5 Performance Comparision

System configuration details:

- Hardware:
  - Workstation : Apple MacPro
  - RAM : 8GB 1066 MHz DDR3
  - Processor : 2 x 2.26 GHz Quad Core Intel Xeon
  - Total no. of cores : 8
  - L2 Cache per core : 256 KB
  - L3 Cache per processor : 8 MB
- Software:
  - Operating System : CentOS release 6.5 (64bit) and Mac OS X version 10.6.8

5.1 Results 1 - BioNetGen vs SSC for the uncoloured benchmarks

Before we compare these two tools with each other, we will quickly go through the criteria and assumptions made for comparison.

Comparison Criteria

The comparison criteria are based on the following parameters:

1. Time taken by a tool for performing simulation.
2. Disk consumption of the result files.

The assumptions and constraints while performing simulation are:

- We are interested in the mean value of the species.
- The interval start will be at 0 time units and the interval end will be 100 time units.
- The simulation algorithm used is Direct/ Gillespie.
- We define an experiment as simulation carried out for a particular value of N and number of Runs.
- Linux command du -sh is used for calculating disk consumption .
- The threshold simulation time for a particular model is 3,600 seconds (1 hour). If the simulation time for a particular model is more than 3,600 seconds (1 hour) then we terminate the simulation.
- No process is running when simulation is being performed.
- For BioNetGen simulation time is displayed by the tool itself at the end, and for SSC date command is used for calculating the time.
- The simulation time for the tools are interpreted in Table 6.

Tools	Read	Simulate	Plot	Write
BioNetGen	Yes	Yes	No	Yes
SSC	Yes	Yes	No	Yes
Snoopy	No	Yes	Yes	No

Table 6: CPU time interpretation of tools

This Table 6 lets us determine the simulation time of a particular tool. e.g. simulation time for BioNetGen includes the reading of the .bngl

file, simulating that .bngl file using the direct/gillespie algorithm as well as writing the result file (ie .cdat, .gdat, net). It does not include the plotting of the curve.

Note- However if you go through the results of each benchmarks in the following pages you may wonder, if this section was about the comparision of BioNetGen and SSC then, why we have Snoopy results lying here. Dear readers, this is because Snoopy is our favourite tool or I should say its my favourite tool. So I just want to see how my tool performs with respect to other two tools. The simulation time (thread 1 is taken into account) taken by Snoopy for each benchmarks are taken directly from [19]. And the size of the result file of Snoopy (i.e. .csv file) is calculated using linux command (du -sh).

Though Snoopy is not the appropiate tool for comparison because these tools are entirely different from each other see Table 6. Also Snoopy has the ability of performing more than one simulation runs and it can give output as the average of it, whereas both BioNetGen and SSC can only perform single simulation run. There is no support for averaging over multiple simulation runs. Also, there's not a simple argument for running multiple SSA simulations. So for example say, if we need to perform simulation 100 times (i.e. run =100) then we have to write a script or think of some other strategy. Now the catch is, if 100 runs are performed the tool will output 100 result files, and not the average of those 100 files. So this requires a large disk space if simulation run is a million times, because it will output a million files. And its we who have to do the averaging.

But, anyways as I said we will still compare Snoopy with these tools, just because I want to know how Snoopy performs and to keep Snoopy in the game.

5.1.1 Benchmark Angiogenesis

Simulation in BioNetGen

The average simulation time and the disk consumption recorded for BioNetGen for this model is given in Table 7 and Table 8 respectively.

N	Run_1	Run_100	Run_10000	Run_1000000
1	0.4	5.0	494.7	> 1hr
5	0.4	5.4	525.7	> 1hr
10	0.4	5.7	541.4	> 1hr
50	0.5	8.1	785	> 1hr

Table 7: BioNetGen, Simulation time (in sec) for Angiogenesis

N	Run_1	Run_100	Run_10000	Run_1000000
1	181 Kb	17.2 Mb	1.72 Gb	> 1hr
5	181 Kb	17.2 Mb	1.72 Gb	> 1hr
10	181 Kb	17.2 Mb	1.72 Gb	> 1hr
50	181 Kb	17.2 Mb	1.72 Gb	> 1hr

Table 8: BioNetGen, Disk Consumption for Angiogenesis

Simulation in SSC

The average simulation time and the disk consumption recorded for SSC for this model is given in Table 9 and Table 10 respectively.

N	Run_1	Run_100	Run_10000	Run_1000000
1	0.003	0.2365	23.6842	2366.5412*
5	0.0047	0.3187	32.0567	3218.6714*
10	0.0059	0.428	43.762	> 1hr
50	0.0150	1.413	142.892	> 1hr

Table 9: SSC,Simulation time (in sec) for Angiogenesis

*Note for ** - The problem with SSC is that, if we perform simulation more than once then for each run a .trj file will be created. So if we have 100 runs then 100 .trj files are created. Then for plotting the graph we need a file which is the average of those 100 files. Since .trj is some sort of binary file therefore we need to convert those files to readable format so that we can take the average of those 100 files. But, converting a file from .trj to .txt format (using ssc-trj-reader-0.01.jar) requires huge amount of time. Just to give you an idea converting 1 angiogenesis .trj file to .txt requires 2 sec, 100

files require 135 sec and 10000 files require 13500 sec. So you can just figure out converting a million .trj file would need (100 x 13500) sec or approx 375 hr which is very large. So for 1 million run simulation was performed, but graph was never plotted.

N	Run_1	Run_100	Run_10000	Run_1000000
1	48 Kb	4.4 Mb	314 Mb	33.2 Gb
5	36 Kb	3.2 Mb	314 Mb	33.2 Gb
10	36 Kb	3.2 Mb	314 Mb	> 1hr
50	36 Kb	3.2 Mb	314 Mb	> 1hr

Table 10: SSC, Disk Consumption for Angiogenesis

Simulation in Snoopy

The average simulation time and the disk consumption recorded for Snoopy for this model is given in Table 11 and Table 12 respectively.

N	Run_1	Run_100	Run_10000	Run_1000000
1	0.0004	0.0317	3.2265	319.9296
5	0.0035	0.3289	31.6073	3169.8615
10	0.0078	0.7567	74.1686	> 1hr
50	0.0476	4.6802	461.9907	> 1hr

Table 11: Snoopy,Simulation time (in sec) for Angiogenesis

N	Run_1	Run_100	Run_10000	Run_1000000
1	8.4 Kb	30.3 Kb	35.9 Kb	34.4 Kb
5	8.4 Kb	32 Kb	33.9 Kb	33 Kb
10	8.5 Kb	31.4 Kb	32.6 Kb	> 1hr
50	9.2 Kb	30.3 Kb	31.3 Kb	> 1hr

Table 12: Snoopy, Disk Consumption for Angiogenesis

Performance comparison

For runtime comparison of the tools refer Figure 10 which is plotted using Table 7 , Table 9 and Table 11

For disk comparison of the tools refer Figure 11 which is plotted using Table 8, Table 10 and Table 12

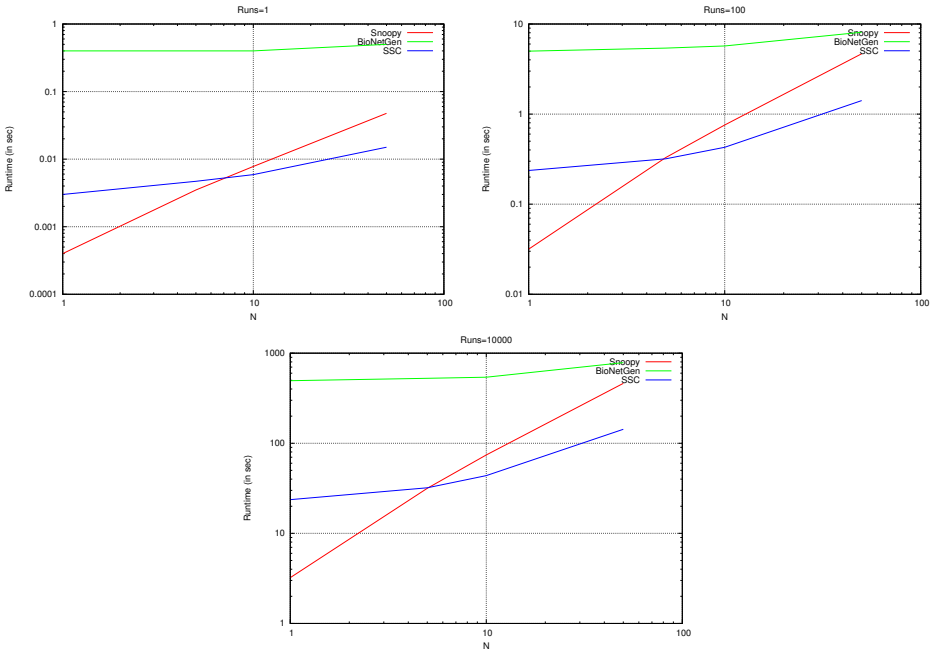

Figure 10: ANGIOGENESIS, Simulation time comparison.

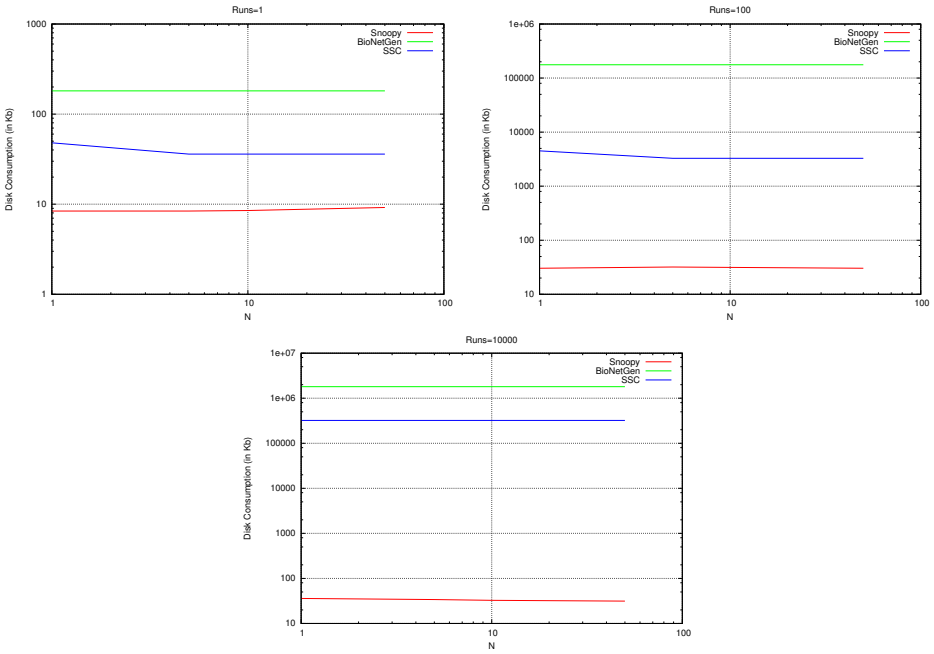

Figure 11: ANGIOGENESIS, Disk Consumption comparison.

5.1.2 Benchmark Erk

Simulation in BioNetGen

The average simulation time and the disk consumption recorded for BioNetGen for this model is given in Table 13 and Table 14 respectively.

N	Run_1	Run_100	Run_10000	Run_1000000
1	0.1	1.9	211.1	> 1hr
100	0.1	3.8	385.6	> 1hr
10000	0.1	6.1	635.7	> 1hr
1000000	4.4	426.7	> 1hr	> 1hr

Table 13: BioNetGen, Simulation time (in sec) for Erk

N	Run_1	Run_100	Run_10000	Run_1000000
1	14 Kb	1.2 Mb	124.9 Mb	> 1hr
100	55 Kb	5.3 Mb	534.5 Mb	> 1hr
10000	55 Kb	5.3 Mb	534.5 Mb	> 1hr
1000000	55 Kb	5.3 Mb	> 1hr	> 1hr

Table 14: BioNetGen, Disk Consumption for Erk

Simulation in SSC

The average simulation time and the disk consumption recorded for SSC for this model is given in Table 15 and Table 16 respectively.

N	Run_1	Run_100	Run_10000	Run_1000000
1	0.003	0.209	20.863	2088.1034*
100	0.003	0.223	22.549	2243.1789*
10000	0.035	3.309	329.206	3304.9832*
1000000	3.037	304.194	> 1hr	> 1hr

Table 15: SSC, Simulation time (in sec) for Erk

*Note for ** - The problem with SSC is that, if we perform simulation more than once then for each run a .trj file will be created. So if we have 100 runs then 100 .trj files are created. Then for plotting the graph we need a file which is the average of those 100 files. Since .trj is some sort of binary file therefore we need to convert those files to readable format so that we can take the average of those 100 files. But, converting a file from .trj to .txt format (using ssc-trj-reader-0.01.jar) requires huge amount of time. Just to give you an idea converting 1 erk .trj file to .txt requires 1 sec, 100 files

require 122 sec and 10000 files require 12235 sec. So you can just figure out converting a million .trj file would need (100 x 12235) sec or approx 340 hrs which is very large. So for 1 million run simulation was performed, but graph was never plotted.

N	Run_1	Run_100	Run_10000	Run_1000000
1	8 Kb	420 Kb	41 Mb	4 Gb
100	16 Kb	1.2 Mb	118 Mb	11.52 Gb
10000	16 Kb	1.2 Mb	118 Mb	11.52 Gb
1000000	16 Kb	1.2 Mb	> 1hr	> 1h

Table 16: SSC, disk Consumption for Erk

Simulation in Snoopy

The average simulation time and the disk consumption recorded for Snoopy for this model is given in Table 17 and Table 18 respectively.

N	Run_1	Run_100	Run_10000	Run_1000000
1	0.0002	0.0018	0.1658	16.8030
100	0.0009	0.0989	8.9830	884.4062
10000	0.0898	8.8005	875.7929	> 1hr
1000000	9.0178	889.899	> 1hr	> 1hr

Table 17: Snoopy,Simulation time (in sec) for Erk

N	Run_1	Run_100	Run_10000	Run_1000000
1	2.6 Kb	9.7 Kb	10.6 Kb	10.3 Kb
100	3.2 Kb	9.0 Kb	9.0 Kb	9.0 Kb
10000	5.3 Kb	8.9 Kb	8.9 Kb	> 1hr
1000000	7.4 Kb	8.3 Kb	> 1hr	> 1hr

Table 18: Snoopy, Disk Consumption for Erk

Performance comparison

For runtime comparison of the tools refer Figure 12 which is plotted using Table 13 , Table 15 and Table 17

For disk comparison of the tools refer Figure 13 which is plotted using Table 14, Table 16 and Table 18

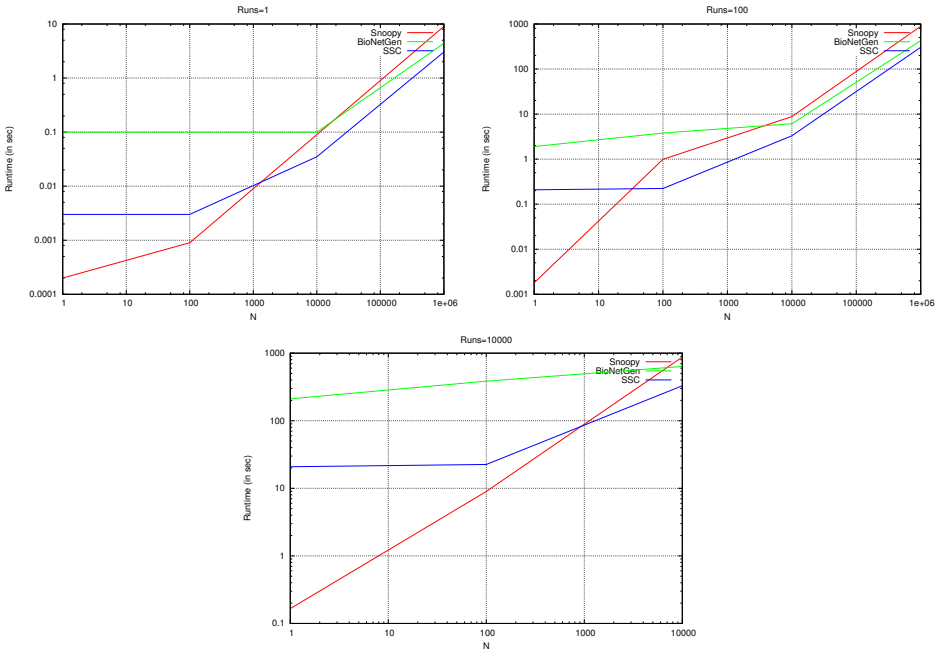

Figure 12: ERK, Simulation time comparison.

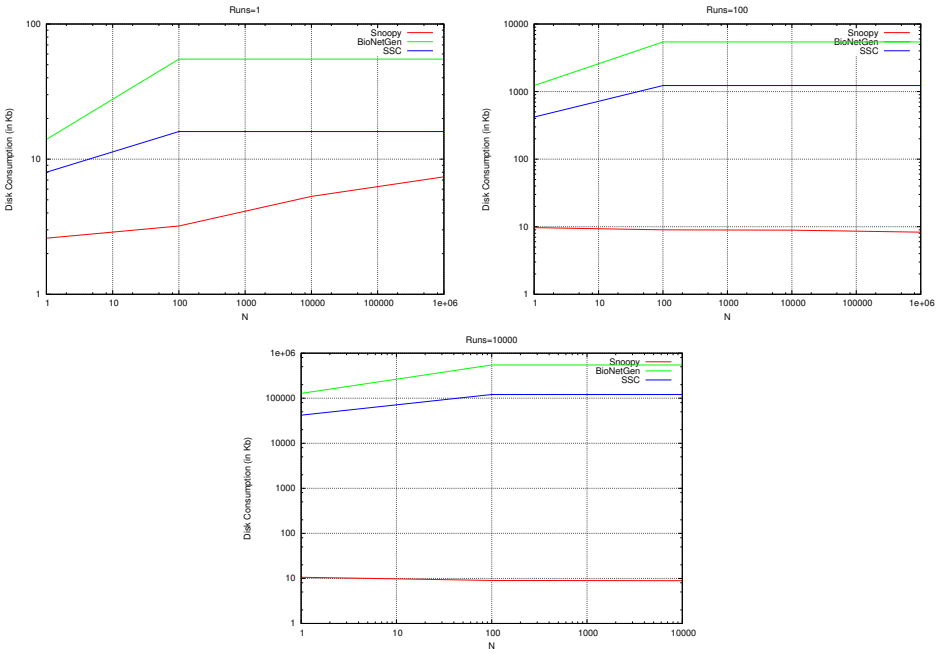

Figure 13: ERK, Disk Consumption comparison.

5.1.3 Benchmark Levchenko

Simulation in BioNetGen

The average simulation time and the disk consumption recorded for BioNetGen for this model is given in Table 19 and Table 20 respectively.

N	Run_1	Run_100	Run_10000	Run_1000000
1	0.2	2.7	278.7	> 1hr
10	0.2	3.2	309.2	> 1hr
100	0.2	3.6	362.3	> 1hr
1000	0.2	9.0	910.2	> 1hr

Table 19: BioNetGen, simulation time (in sec) for Levchenko

N	Run_1	Run_100	Run_10000	Run_1000000
1	107 Kb	10.2 Mb	1.02 Gb	> 1hr
10	111 Kb	10.6 Mb	1.06 Gb	> 1hr
100	111 Kb	10.6 Mb	1.06 Gb	> 1hr
1000	111 Kb	10.6 Mb	1.06 Gb	> 1hr

Table 20: BioNetGen, disk Consumption for Levchenko

Simulation in SSC

The average simulation time and the disk consumption recorded for SSC for this model is given in Table 21 and Table 22 respectively.

N	Run_1	Run_100	Run_10000	Run_1000000
1	0.003	0.222	21.888	2213.1464*
10	0.003	0.223	23.666	2298.6547*
100	0.006	0.464	46.874	> 1hr
1000	0.029	2.777	280.324	> 1hr

Table 21: SSC, simulation time (in sec) for Levchenko

*Note for ** - The problem with SSC is that, if we perform simulation more than once then for each run a .trj file will be created. So if we have 100 runs then 100 .trj files are created. Then for plotting the graph we need a file which is the average of those 100 files. Since .trj is some sort of binary file therefore we need to convert those files to readable format so that we can take the average of those 100 files. But, converting a file from .trj to .txt format (using ssc-trj-reader-0.01.jar) requires huge amount of time. Just to give you an idea converting 1 levchenko .trj file to .txt requires 2 sec, 100

files require 127 sec and 10000 files require 12826 sec. So you can just figure out converting a million .trj file would need (100 x 12826) sec or approx 356 hrs which is very large.So for 1 million run simulation was performed, but graph was never plotted.

N	Run_1	Run_100	Run_10000	Run_1000000
1	24 Kb	1.9 Mb	188 Mb	18.35 Gb
10	24 Kb	1.9 Mb	188 Mb	18.3 Gb
100	24 Kb	2.0 Mb	196 Mb	> 1hr
1000	24 Kb	2.0 Mb	196 Mb	> 1hr

Table 22: SSC, disk Consumption for Levchenko

Simulation in Snoopy

The average simulation time and the disk consumption recorded for Snoopy for this model is given in Table 23 and Table 24 respectively.

N	Run_1	Run_100	Run_10000	Run_1000000
1	0	0.0105	1.0379	104.4715
10	0.0011	0.1114	11.3115	1066.5433
100	0.0115	1.1429	112.4734	> 1hr
1000	0.1166	11.3270	1152.1005	> 1hr

Table 23: Snoopy,Simulation time (in sec) for Levchenko

N	Run_1	Run_100	Run_10000	Run_1000000
1	4.8 Kb	17.6 Kb	19.6 Kb	19.3 Kb
10	5.4 Kb	17.2 Kb	18.3 Kb	18.4 Kb
100	6.7 Kb	17.0 Kb	17.9 Kb	> 1hr
1000	8.3 Kb	17.0 Kb	17.5 Kb	> 1hr

Table 24: Snoopy, Disk Consumption for Levchenko

Performance comparison

For runtime comparison of the tools refer Figure 14 which is plotted using Table 19 , Table 21 and Table 23

For disk comparison of the tools refer Figure 15 which is plotted using Table 20, Table 22 and Table 24

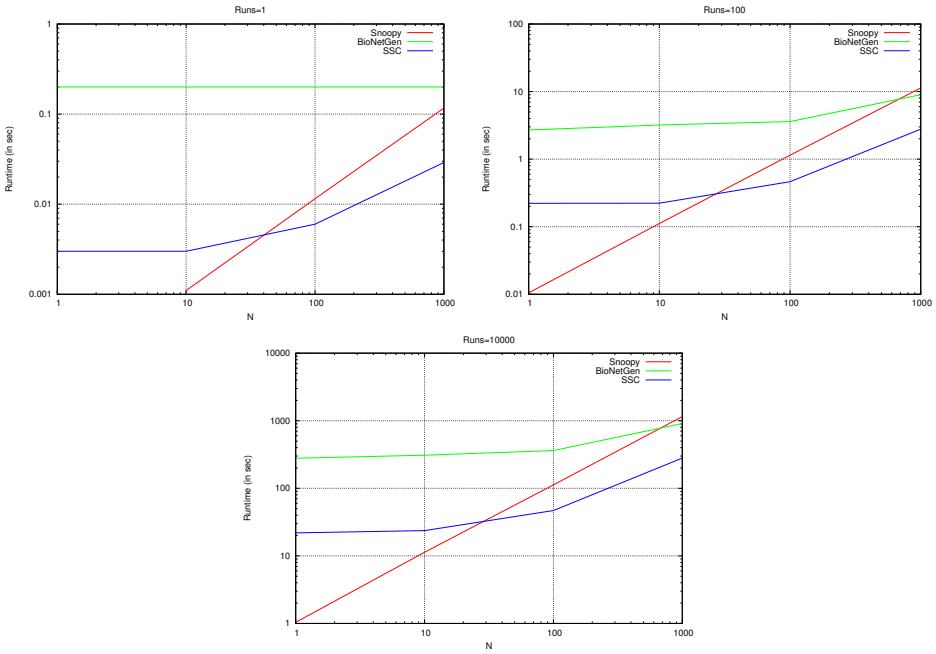

Figure 14: Levchenko, Simulation time comparison.

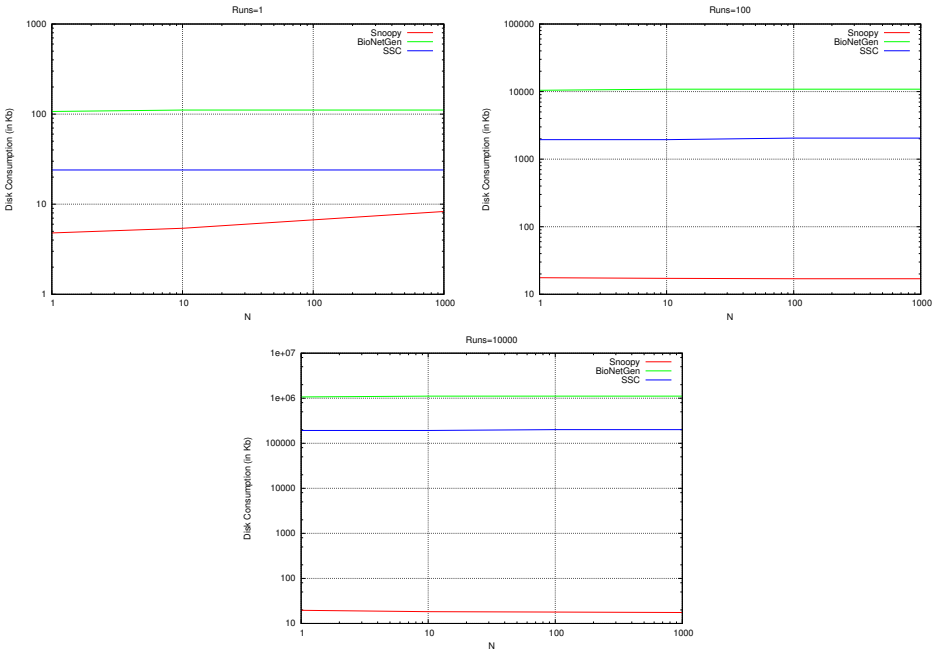

Figure 15: LEVCHENKO, Disk Consumption comparison.

5.1.4 Conclusion

So if look all the previous graphs, we can say that, for

Simulation time

- If we compare BioNetGen and SSC, then SSC is always a clear winner i.e. SSC takes less time in doing simulation.
- Snoopy turns out to be the fastest if, N is small.
- However, it loses to SSC as the value of N increases, but then also it performs better than BioNetGen.

Disk Consumption

For this we always have a clear picture i.e. Snoopy consumes least amount of disk space whereas, BioNetGen consumes the most. Therefore the ranking are as follows -

- Snoopy
- SSC
- BioNetGen

However the results are as expected, because both BioNetGen and SSC output the result file for each run rather than giving one averaged result file. Hence the ranking.

5.2 Results 2 - Cain, Marcie, Snoopy and Stochkit for the coloured benchmarks

In this section, the remaining tools, i.e. CAIN, Marcie, Snoopy and StochKit are compared with each other.

So the first thing that strikes your mind is why I am being so biased. Why we have two results section one for BioNetGen and SSC and another for the remaining tools? Why ?

- This is because, I was asked to extend the work done by Aman Sinha [19]. And if you go through his report he didn't have BioNetGen and SSC.
- BioNetGen and SSC are very different from these four tools. You cannot compare a donkey with a horse or a goat with a sheep. I mean if you want you can, but my point is, they are entirely different from each other. BioNetGen and SSC have no averaging facility. They will output 100 result files for each run (if runs =100) whereas, these four tools will give you one final averaged file.
- Also, there is no scheme of threads available in BioNetGen and SSC.
- Further, you can refer the table Table 6 and Table 25. So now you can see they lie on opposite poles. Hence the two sections.

The comparison criteria are based on the following parameters:

1. Simulation Run-time comparision
2. Memory consumption comparison

The assumptions and constraints while performing simulation are:

- We are interested in the mean value of the species.
- When simulations are being performed no other processes should be running.
- For benchmark Repressilator the interval start will be at 0 time units and the interval end will be 10000 time units.
- The simulation algorithm used is Direct/ Gillespie.
- Threads used will be 1, 4, 8 and 16 which will be mentioned explicitly.

- We define an experiment as simulation carried out for a particular value of N, number of runs and threads.
- A total of 10 trials is performed for each experiment and for each benchmark model.
- After performing 10 trials, the mean value of the simulation runtime and the corresponding peak memory consumption is calculated.
- The threshold simulation runtime for a particular model is 3,600 seconds (1 hour). If the simulation runtime for a particular model is more than 3,600 seconds (1 hour) then we terminate the simulation.
- The memory is calculated using a shell script. It calculates memory consumption in KB and has a sampling time of 0.1 seconds.
- For a GUI tool in-order to calculate the memory consumption the tool has to be reopened before each experiment.
- For unfolding the coloured petri net, thread 8 is used.
- 10 trials are carried out keeping the scaling parameter, no of runs and no of threads constant.
- Simulation runtime for each trial is noted.
- The memory consumption for each trial is also recorded and the maximum/peak memory consumption is taken into account.
- The runtime of each trial is recorded. Such 10 trails are recorded and the average runtime is calculated. The average runtime calculated is the runtime of a particular experiment.
- This average runtime and the peak memory consumption is the simulation runtime and memory consumption of an experiment respectively.
- While performing simulation on CAIN the granularity and priority sliders are kept to their default value.
- While performing simulation on Marcie only the total elapsed time is noted. The total elapsed time is the runtime of the simulation.
- Stochkit uses a SSA driver for performing stochastic simulation. It selects appropriate simulation method to achieve the best performance. For more details see [17] and refer StochKit manual.

Tools	Read	Simulate	Plot	Write
Cain	No	Yes	No	No
Marcie	Yes	Yes	No	Yes
Snoopy	No	Yes	Yes	No
Stochkit	No	Yes	No	No

Table 25: CPU time interpretation of tools

The runtime of the tools are interpreted in the Table 25:

The above table means that the simulation runtime of a particular tool is determined by the above steps. e.g. simulation runtime for snoopy includes the simulation time of the direct/gillespie algorithm as well as plotting of the curve. It doesn't include the reading time of the SBML file and the time spend in writing the result into a file (in this case .csv file).

In case of Marcie the total processing time includes time for reading, simulation and writing. However we are only interested in the time for simulation. The simulation time displayed in Marcie is the total elapsed time. We record the total elapsed time for the experiments.

5.2.1 Benchmark Gradient

Note- Before we start with Gradient, I have a confession to make. I had very limited time, and thats the reason why, I could not complete the simulations with Cain and Snoopy. So for gradient you will not find the results of simulation time and peak memory comparision with Cain and Snoopy. However, we have tried to compare Marcie and StochKit with each other.

Simulation in Marcie

The average runtime and the peak memory consumption recorded for Marcie for this model is given in Table 26 and Table 27 respectively.

N	Threads	Runs_1	Runs_100	Runs_1000	Runs_10000
10	1	0	19.5	194.6	1948.8
	4	0	4.5	49.4	498.7
	8	0	3	26.7	271.3
	16	0	3	24.2	232
50	1	2	143.3	1415.5	$>$ 1 hr
	4	2	36.3	360.8	3574.8
	8	2	24	188.4	1871.4
	16	2	20.3	162.2	1537.3

Table 26: MARCIE, average runtime (in sec) for Gradient.

N	Threads	Runs_1	Runs_100	Runs_1000	Runs_10000
10	1	5060	5092	5096	5096
	4	5060	8520	8548	8560
	8	5060	10428	10452	10472
	16	5060	14236	14264	16312
50	1	58000	58912	60036	$>$ 1 hr
	4	58000	95580	97688	97600
	8	58000	142980	147048	144540
	16	58000	237692	237748	239496

Table 27: MARCIE peak memory consumption (in KB) for Gradient.

Simulation in StochKit

The average runtime and the peak memory consumption recorded for StochKit for this model is given in Table 28 and Table 29 respectively.

N	Threads	Runs_1	Runs_100	Runs_1000	Runs_10000
10	1	0.2615667	18.92915	186.5257	1866.994
	4	0.2477467	5.00963	47.58659	474.4301
	8	0.2542659	2.969362	25.00652	247.0285
	16	0.263761	2.610484	20.83053	203.4524
50	1	4.812265	302.601	3011.427	$>$ 1 hr
	4	4.926437	78.95675	764.2567	$>$ 1 hr
	8	4.861952	43.61127	404.6536	$>$ 1 hr
	16	4.842285	33.26937	288.1078	2834.773

Table 28: STOCHKIT, average runtime (in sec) for Gradient.

N	Threads	Runs_1	Runs_100	Runs_1000	Runs_10000
10	1	4668	4672	4692	4696
	4	4668	4868	6556	6556
	8	4664	6932	6936	6936
	16	4668	9080	9080	9080
50	1	56004	56004	56004	$>$ 1 hr
	4	56004	56196	56004	$>$ 1 hr
	8	56004	56240	56004	$>$ 1 hr
	16	56004	62468	58384	58384

Table 29: STOCHKIT peak memory consumption (in KB) for Gradient.

Performance comparison

For runtime comparison of the tools refer Figure 16, Figure 17, Figure 18 and Figure 19 which is plotted using Table 26 , Table 28

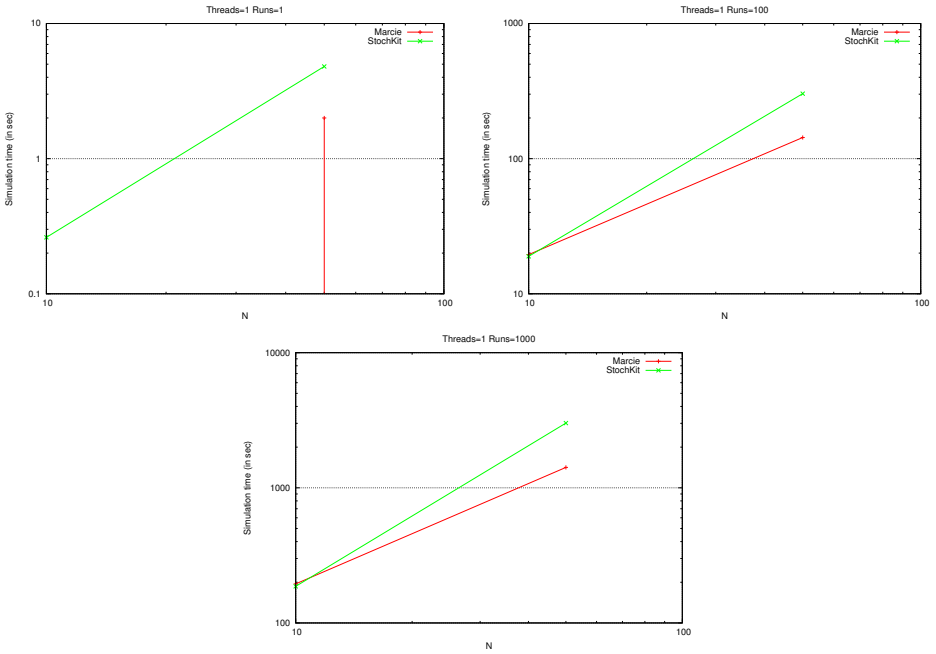

Figure 16: Gradient, Simulation time comparison for Thread=1.

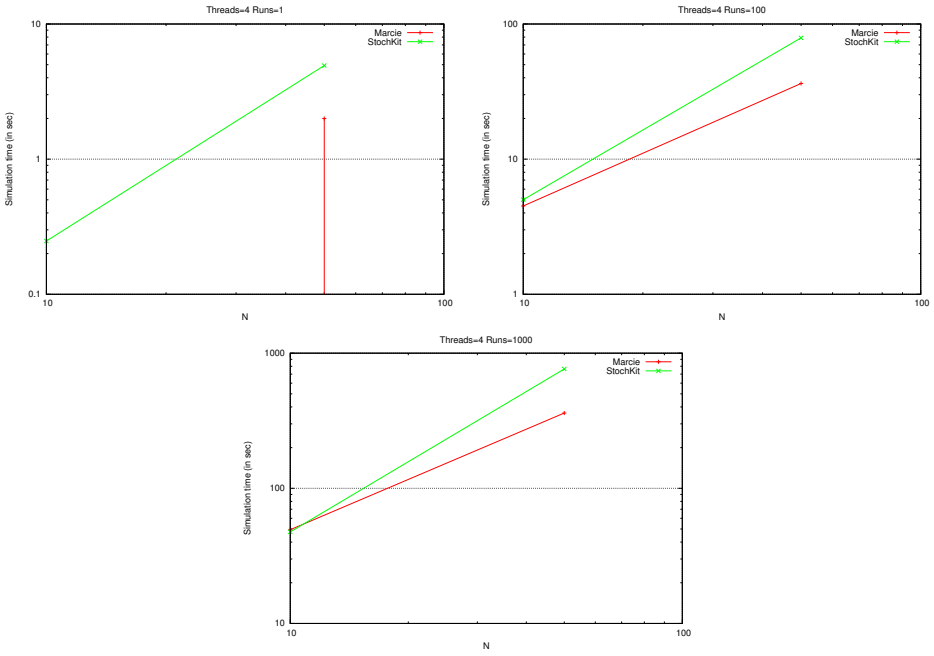

Figure 17: Gradient, Simulation time comparison for Thread=4.

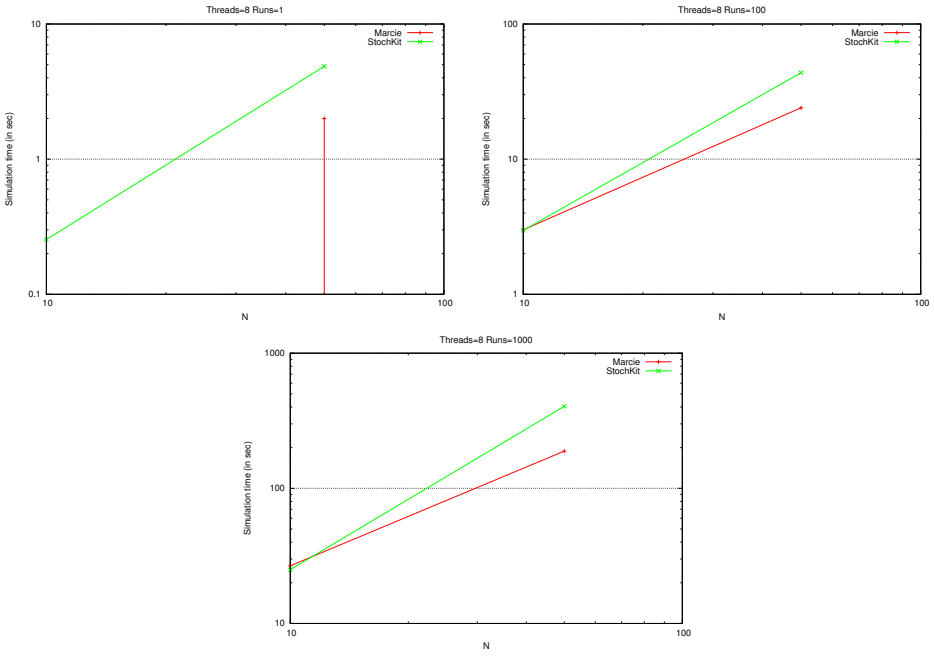

Figure 18: Gradient, Simulation time comparison for Thread=8.

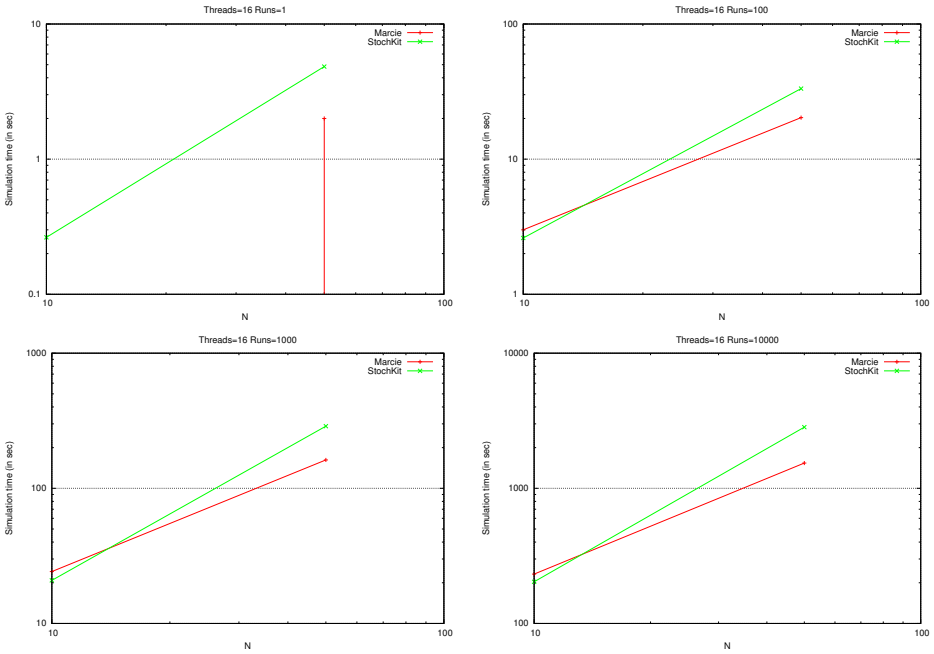

Figure 19: Gradient, Simulation time comparison for Thread=16.

And, for peak memory consumption of the tools refer Figure 20, Figure 21, Figure 22 and Figure 23 which is plotted using Table 27 and Table 29

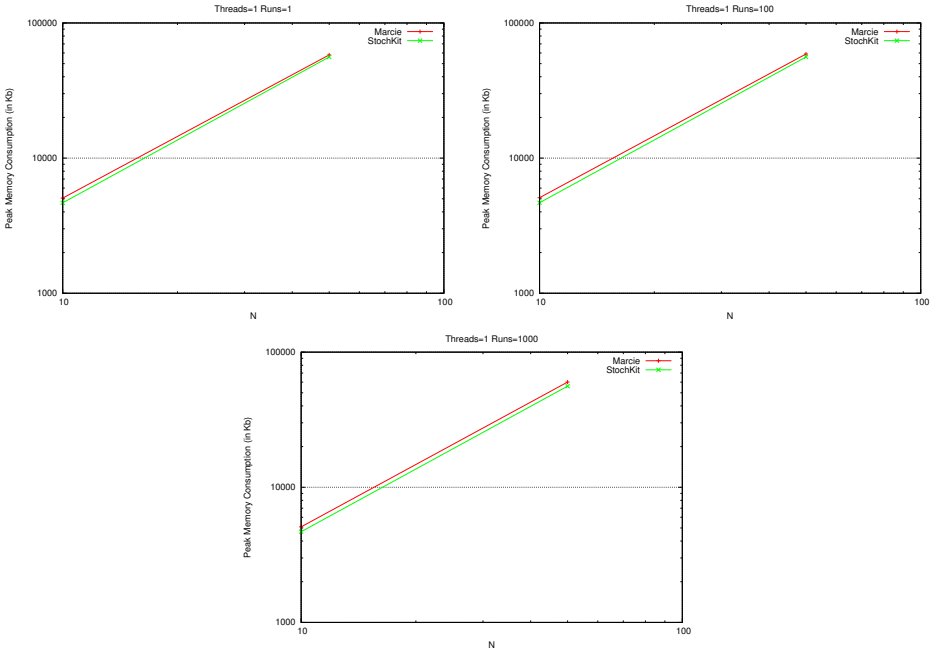

Figure 20: Gradient, Peak Memory comparison for Thread=1.

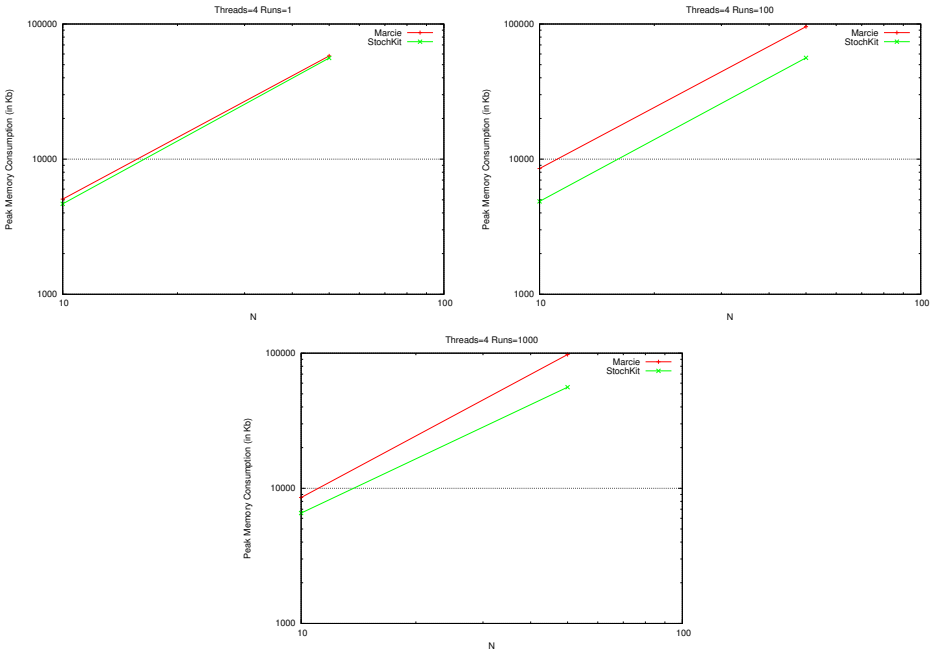

Figure 21: Gradient, Peak Memory comparison for Thread=4.

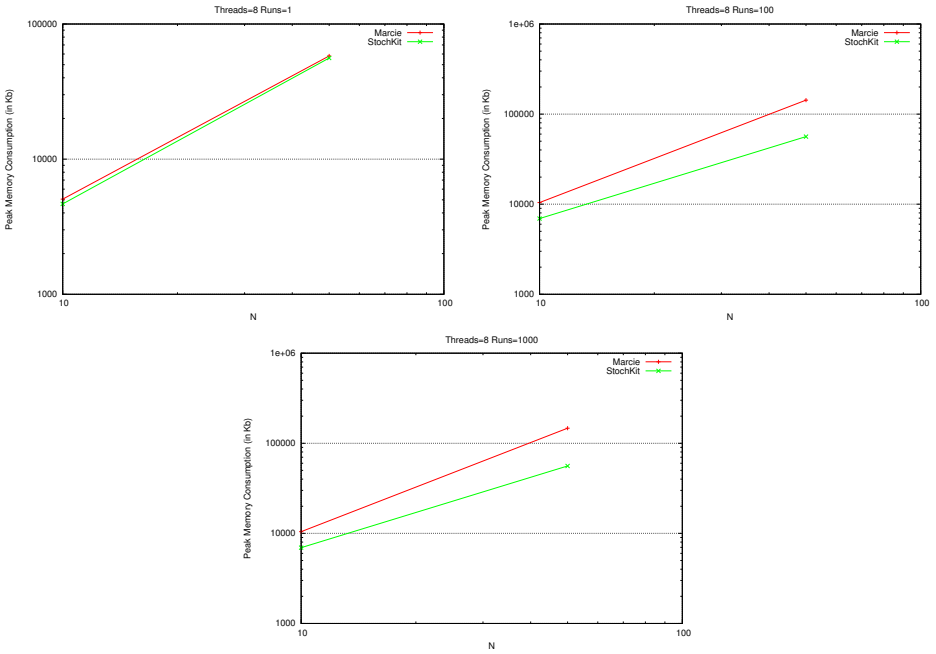

Figure 22: Gradient, Peak Memory comparison for Thread=8.

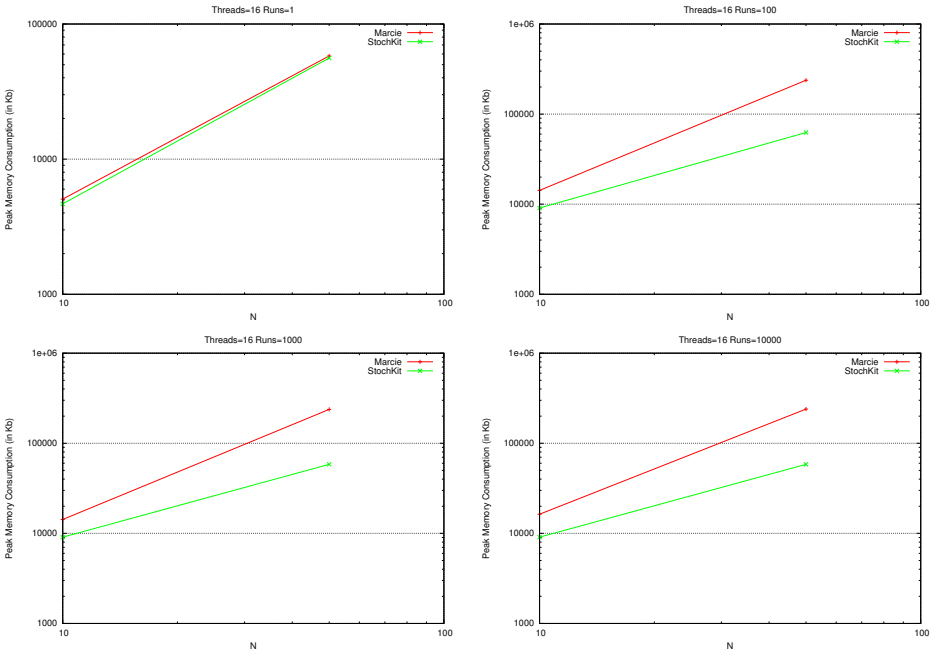

Figure 23: Gradient, Peak Memory comparison for Thread=16.

5.2.2 Benchmark Repressilator

Simulation in Cain

The average runtime and the peak memory consumption recorded for Cain for this model is given in Table 30 and Table 31 respectively.

Note -For Table 30 and Table 31, * signifies that either Cain crashes during simulation or it crashes while exporting data to .csv format.

N	Threads	Runs_1	Runs_100	Runs_1000	Runs_10000
3	1	0.0227811	0.1029339	0.7970706	7.4205718
	4	0.0231064	0.0870712	0.4392889	3.675791
	8	0.0233402	0.1232316	0.473155	3.787557
	16	0.0227331	0.210041667	0.589227	4.0476
30	1	0.027182	0.751430333	7.127259	69.606281
	4	0.026804333	0.374263333	2.852957667	27.68598067
	8	0.027535333	0.390929333	2.952203	28.14462533
	16	0.028904333	0.529854667	3.132413	28.87133467
100	1	0.06481	2.598667	25.10264333	248.4859203
	4	0.065613333	1.112333667	9.663518	94.14641733
	8	0.064176333	1.169748	9.772885667	95.089383
	16	0.064053333	1.460976333	10.089256	96.282014
300	1	0.231117333	8.604342667	82.39522033	821.9003437
	4	0.240179	4.161969	37.11558467	*
	8	0.230506333	4.299662333	35.51482167	*
	16	0.234675	4.944599333	35.70754067	*
3000	1	11.19500667	262.6925763	*	*
	4	11.46225467	107.7790243	*	*
	8	11.25181633	124.2901567	*	*
	16	11.18200933	167.562019	*	*

Table 30: CAIN, average runtime (in sec) for Repressilator.

N	Threads	Runs_1	Runs_100	Runs_1000	Runs_10000
3	1	108832	110528	118368	185900
	4	110872	115700	122168	188196
	8	109000	120076	128124	196812
	16	114752	121948	142768	211036
30	1	111872	118232	180704	818036
	4	111892	123496	187264	820320
	8	109804	131952	196384	828104
	16	111800	145132	213224	842352
100	1	113200	139316	349136	2463932
	4	114680	147084	355168	2471004
	8	114680	159860	370076	2477624
	16	114704	252748	396852	2496256
300	1	125620	197292	832008	35572996
	4	121984	221352	847688	*
	8	124044	243052	864436	*
	16	126964	287344	924780	*
3000	1	276732	1031784	*	*
	4	276668	1200552	*	*
	8	276828	1407192	*	*
	16	276832	1668128	*	*

Table 31: CAIN peak memory consumption (in KB) for Repressilator.

Simulation in Marcie

The average runtime and the peak memory consumption recorded for Marcie for this model is given in Table 32 and Table 33 respectively.

N	Threads	Runs_1	Runs_100	Runs_1000	Runs_10000
3	1	0	0	0	4
	4	0	0	0	1
	8	0	0	0	0
	16	0	0	0	0
30	1	0	0	6	60.4
	4	0	0	1	15
	8	0	0	0	7.1
	16	0	0	0	6.4
100	1	0	2	22	225.6
	4	0	0	6	66.3
	8	0	0	3	31.2
	16	0	0	2	22
300	1	0	7.1	77.2	764.9
	4	0	2	19.2	194.5
	8	0	1	9	97.1
	16	0	1	8	75.2
3000	1	2	126.2	1207.4	$>$ 1hr
	4	2	33.4	312.7	3032.3
	8	2	21.7	155.4	1529.1
	16	2	17.1	123.3	1117.8

Table 32: Marcie, average runtime (in sec) for Repressilator.

N	Threads	Runs_1	Runs_100	Runs_1000	Runs_10000
3	1	3284	3284	3284	3280
	4	3284	5344	9424	5340
	8	3284	5552	5552	5548
	16	3284	5952	5960	5956
30	1	4172	4172	4172	4196
	4	4172	6884	6884	6908
	8	4168	7948	7956	7976
	16	4172	10092	10096	10161.5
100	1	6312	6316	8352	6344
	4	6312	10852	10832	10868
	8	6312	14124	14132	14248
	16	6312	20720	22764	20768
300	1	12832	12836	14868	12872
	4	12832	22124	22148	22152
	8	12832	31788	31820	31832
	16	12832	51128	51152	51180
3000	1	100636	101844	102524	$>$ 1hr
	4	100636	174384	176340	176364
	8	100636	270036	270092	272016
	16	100636	461372	461384	462944

Table 33: Marcie peak memory consumption (in KB) for Repressilator.

Simulation in Snoopy

The average runtime and the peak memory consumption recorded for Snoopy for this model is given in Table 34 and Table 35 respectively.

N	Threads	Runs_1	Runs_100	Runs_1000	Runs_10000
3	1	0.0007	0.0584	0.5725	5.5719
	4	1.0016	1.0064	1.0067	2.0066
	8	1.0015	1.0135	1.0126	1.2132
	16	1.0017	1.0254	1.025	7.4262
30	1	0.0073	0.7119	7.0995	70.8236
	4	1.002	1.006	2.0064	18.00878
	8	1.0018	1.0146	1.4131	10.0137
	16	1.0017	1.0257	1.0258	7.4262
100	1	0.0268	2.6495	26.4528	264.1104
	4	1.002	1.0074	7.5076	69.5144
	8	1.0016	1.0141	4.1147	36.7179
	16	1.0019	1.0268	3.1278	27.6283
300	1	0.0935	9.1666	89.3235	881.4462
	4	1.0024	3.0074	22.9105	230.7623
	8	1.0021	2.0158	13.7156	119.8251
	16	1.0024	2.0292	10.0295	90.3363
3000	1	1.506	146.8005	1359.6467	$>$ 1hr
	4	2.0038	38.219	367.358	3341.7593
	8	2.0038	25.8315	202.1269	1806.2736
	16	2.004	19.656	152.8652	1331.4382

Table 34: Snoopy, average runtime (in sec) for Repressilator.

N	Threads	Runs_1	Runs_100	Runs_1000	Runs_10000
3	1	37336	39020	39132	37156
	4	38236	38316	38464	38404
	8	37248	37560	40672	40688
	16	37212	39096	43176	43268
30	1	40820	40008	39504	39000
	4	38960	40896	40532	41008
	8	39160	43712	43580	42222
	16	38916	45712	48072	48488
100	1	44988	45800	43792	45852
	4	44028	46620	48236	49320
	8	45188	51992	51964	53580
	16	45856	61008	60920	62414
300	1	61364	60324	58232	60256
	4	59532	63716	68324	68540
	8	58532	75580	80336	80312
	16	61768	97088	100056	100020
3000	1	284300	252656	256192	$>$ 1hr
	4	252408	341496	340196	340164
	8	252704	439116	438988	440604
	16	255688	633172	633260	635688

Table 35: Snoopy peak memory consumption (in KB) for Repressilator.

Simulation in Stochkit

The average runtime and the peak memory consumption recorded for StochKit for this model is given in Table 36 and Table 37 respectively.

N	Threads	Runs_1	Runs_100	Runs_1000	Runs_10000
3	1	0.009098244	0.05042284	0.4172069	4.080088
	4	0.00940475	0.02780769	0.1202634	1.044103
	8	0.009073304	0.03137051	0.08045206	0.5418112
	16	0.009230327	0.0476595	0.08379927	0.4242532
30	1	0.03096062	0.7450247	7.214039	71.0286
	4	0.03173159	0.2252811	1.846212	18.03295
	8	0.03026337	0.1527049	1.001117	9.130265
	16	0.03125842	0.1593174	0.8036006	7.165044
100	1	0.1058777	3.973354	38.17591	380.8624
	4	0.1062088	1.117741	10.010083	98.28253
	8	0.1113469	0.678797	5.073894	49.23492
	16	0.1113071	0.689728	4.343678	41.51228
300	1	0.5148289	24.13415	236.8992	2370.306
	4	0.5194753	6.748251	61.57599	608.1067
	8	0.5177906	3.730049	31.2567	304.4749
	16	0.5128759	3.934516	28.75998	277.4131
3000	1	9.992275	170.4117	1572.279	$>$ 1hr
	4	9.969124	50.35427	418.8726	$>$ 1hr
	8	9.974653	31.85645	234.1409	2153.288
	16	9.999077	34.01938	222.2411	2036.905

Table 36: Stochkit, average runtime (in sec) for Repressilator.

N	Threads	Runs_1	Runs_100	Runs_1000	Runs_10000
3	1	3020	3020	3020	3020
	4	3020	4936	4936	4936
	8	3020	7020	4984	7024
	16	3020	5076	9172	9184
30	1	3612	3612	3616	3648
	4	3612	4996	4996	5592
	8	3648	7080	9124	7676
	16	3648	9236	9792	9236
100	1	5432	5436	5440	5464
	4	5436	7280	7280	7280
	8	5436	9368	9364	7328
	16	5436	11484	11488	13528
300	1	10588	10592	10612	10612
	4	10588	12172	12172	16248
	8	10588	14260	12220	14260
	16	10588	16408	18448	16404
3000	1	81564	81564	81560	$>$ 1hr
	4	81564	89108	91060	$>$ 1hr
	8	81564	89052	88936	89052
	16	81564	89160	89088	89144

Table 37: Stochkit peak memory consumption (in KB) for Repressilator.

Performance comparison

For runtime comparison of the tools refer Figure 24, Figure 25, Figure 26 and Figure 27 which is plotted using Table 30 , Table 32, Table 34 , Table 36

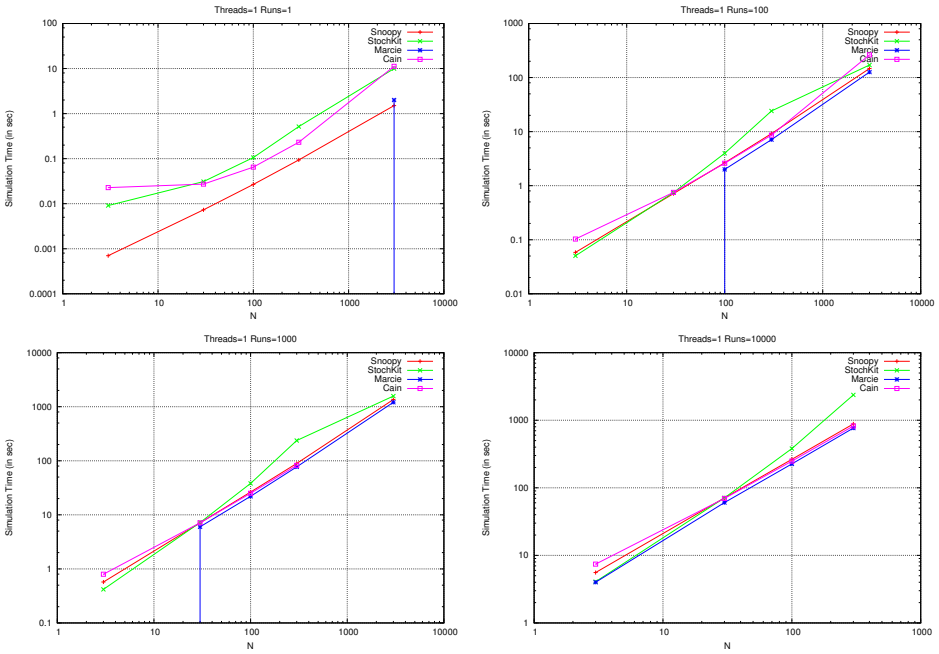

Figure 24: REPRESSILATOR, Simulation time comparison for Thread=1.

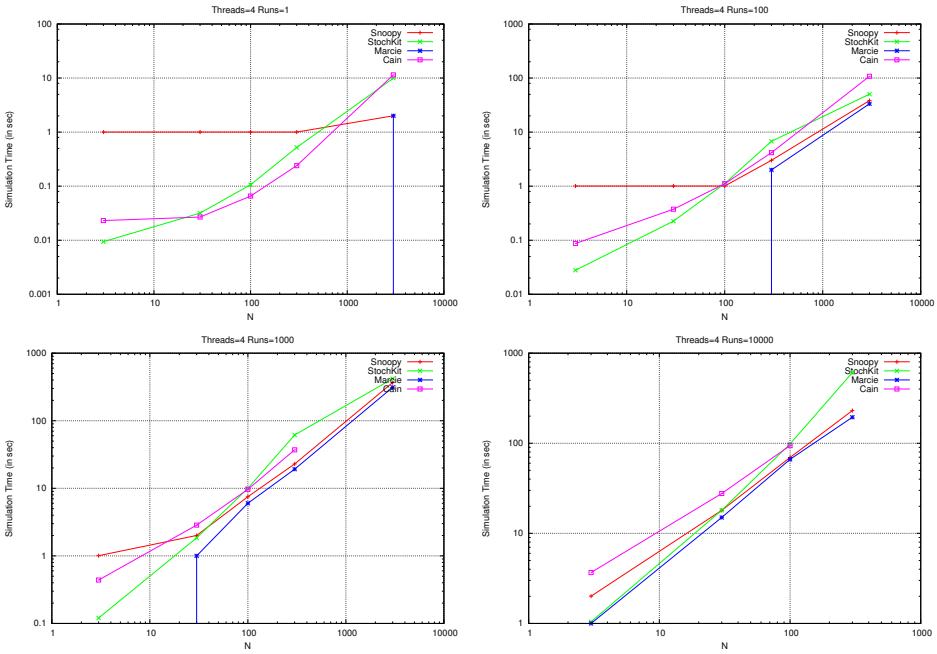

Figure 25: REPRESSILATOR, Simulation time comparison for Thread=4.

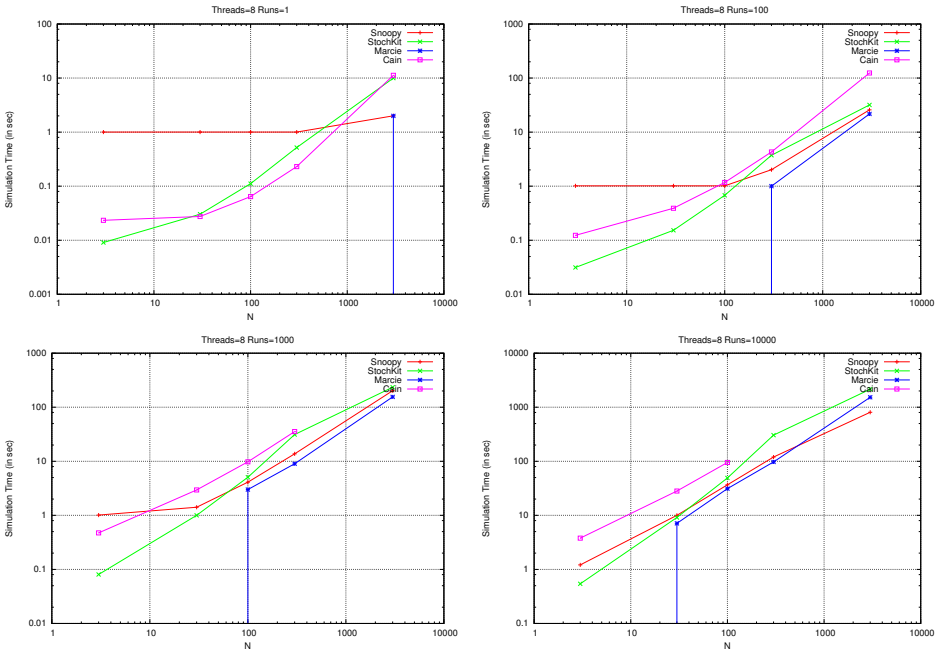

Figure 26: REPRESSILATOR, Simulation time comparison for Thread=8.

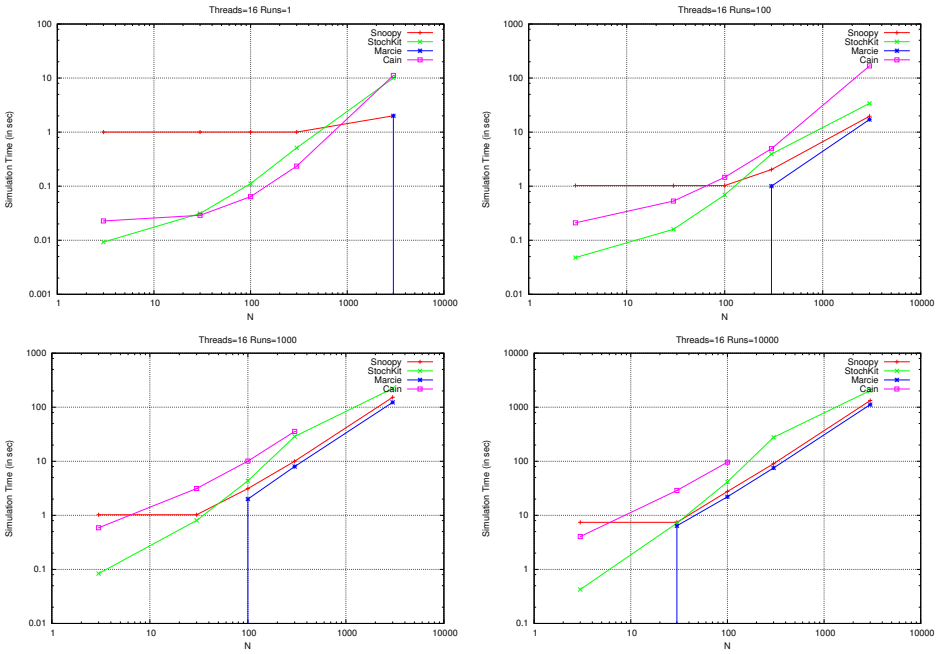

Figure 27: REPRESSILATOR, Simulation time comparison for Thread=16.

And, for peak memory consumption of the tools refer Figure 28, Figure 29, Figure 30 and Figure 31 which is plotted using Table 31, Table 33, Table 35 and Table 37

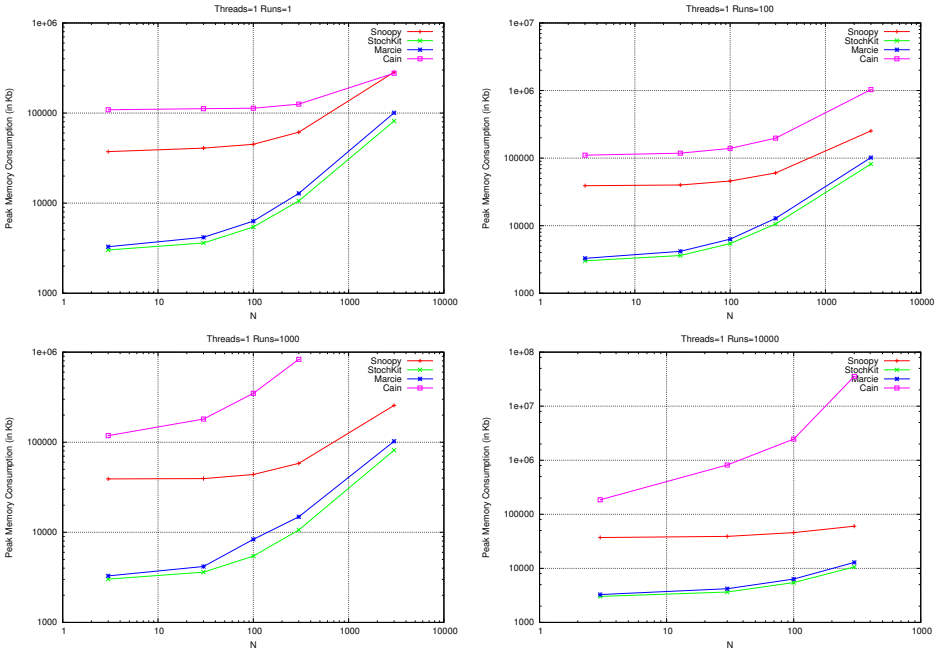

Figure 28: Repressilator, Peak Memory comparison for Thread=1.

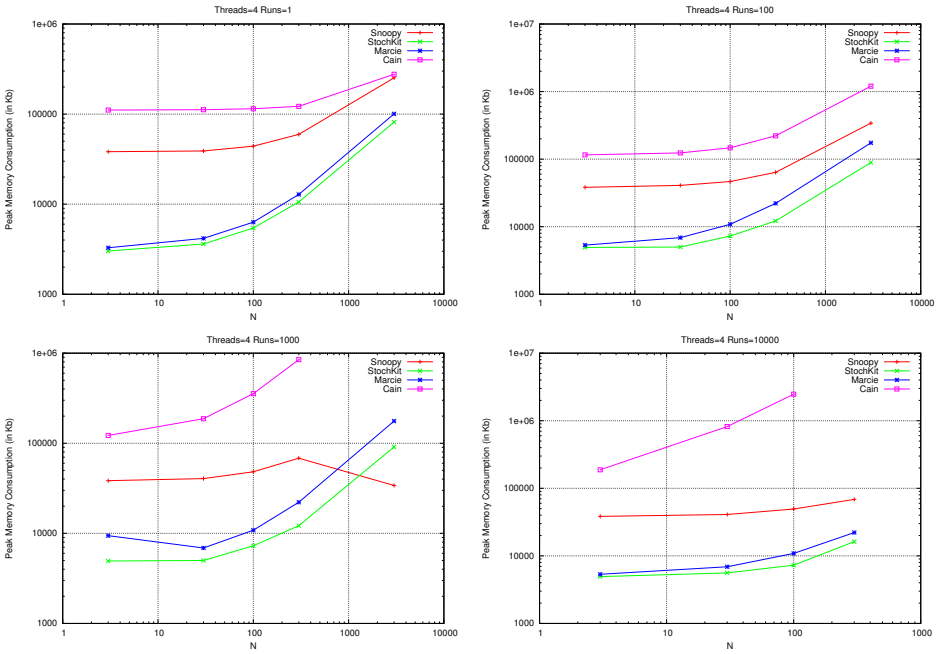

Figure 29: Repressilator, Peak Memory comparison for Thread=4.

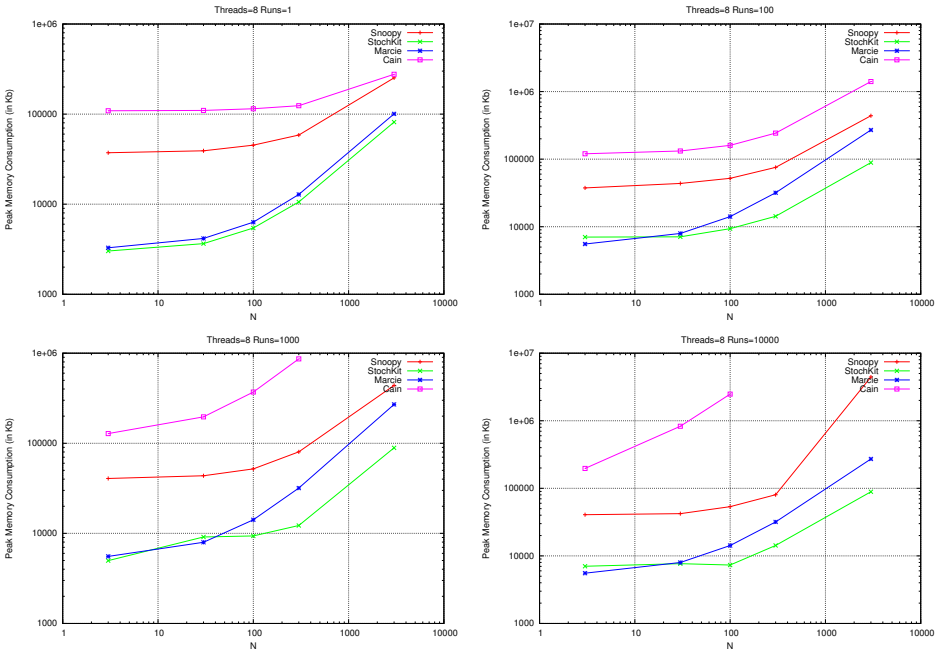

Figure 30: Repressilator, Peak Memory comparison for Thread=8.

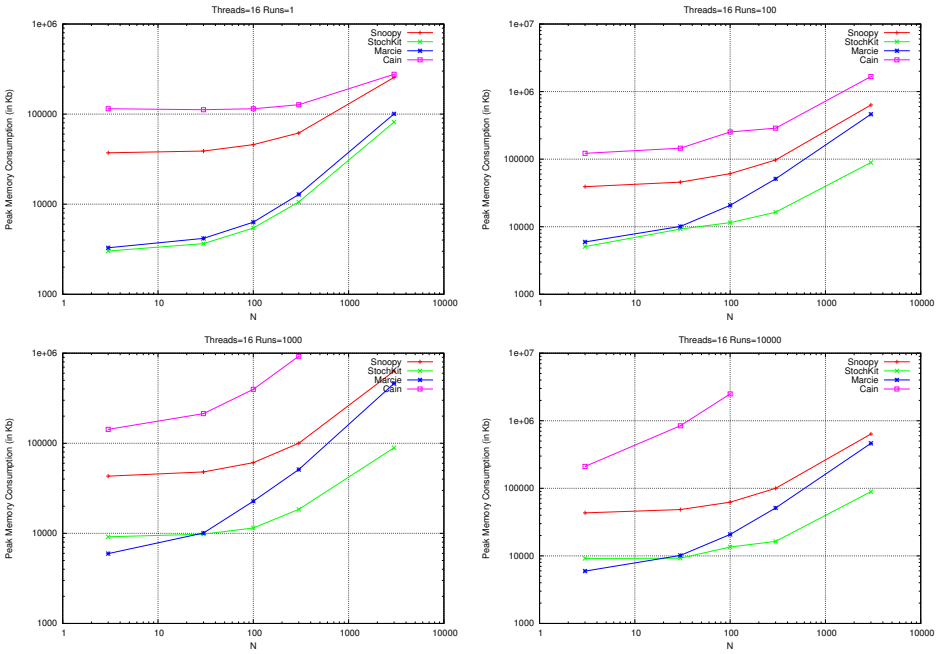

Figure 31: Repressilator, Peak Memory comparison for Thread=16.

5.2.3 Conclusion

Gradient Benchmark
For Simulation time, though marcie gives a bad start at first, but at the later stages it turns out to be the clear champion i.e. Marcie takes less time than StochKit in performing simulation. However it has to compromise memory for it.

For memory comparison, for run =1 theres no much difference between the two tools, however for run >1 we see that Stochkit consumes less memory than Marcie.

Repressilator Benchmark

For Simulation time, its very difficult to decide, but in general we can say that for greater value of N, the order is
Marcie < Snoopy < StochKit < Cain.

For less value of N, its very difficult to decide.

For memory comparison, the order is,
StochKit < Marcie < Snoopy < Cain
which means cain requires the highest amount of memory and StochKit the least.

6 Summary

6.1 Achievements

- Before the start of this project we did not had an export from stochastic Petri nets to SBML Level 1. Also there was no export for coloured Petri nets to SBML level 1 and level 2. We successfully added these features to Snoopy.
- We performed comparision between BioNetGen and SSC with three benchmarks, i.e. Angiogenesis, Erk and Levchenko.
- We also extended the work done by Aman Sinha [19]. In this very limited time, we added two benchmarks to it, i.e. Gradient and Repressilator, which from my point of view is highly appreciable.
- The report is written in LaTeX which was completely new to me. So yes, I learned a few basics about LaTeX, too.

6.2 Open Problems

There are a few potential areas where this work can be extended. Some of them are:

- As the size of benchmark increases, it becomes increasingly important that we carry out simulations on more powerful machines.
- Instead of relying upon third party app for conversion from SBML to bngl, we can try writing our own export.
- We can add more benchmarks for comparing SSC and BioNetGen.
- In a similar fashion, more benchmarks could be added for Cain, Marcie, Snoopy and Stochkit.

References

[1] R. Blossey, L. Cardelli, and A. Phillips. Compositionality, Stochasticity and Cooperativity in Dynamic Models of Gene Regulation. *HFSP Journal*, 1(2):17–28, 2008.

[2] J. Bosak and Bray. Xml and the second-generation web. Technical Report 280(5):8993., Scientific American, 1999.

[3] F. Cordero, A. Horváth, D. Manini, L. Napione, M. D. Pierro, S. Pavan, A. Picco, A. Veglio, M. Sereno, F. Bussolino, and G. Balbo. Simplification of a complex signal transduction model using invariants and flow equivalent servers. *Theor. Comput. Sci.*, 412(43):6036–6057, 2011.

[4] D. Gilbert and M. Heiner. *From Petri nets to differential equations - an integrative approach for biochemical network analysis*, pages 181–200. LNCS 4024, Springer, 2006.

[5] D. Gilbert, M. Heiner, and S. Lehrack. A unifying framework for modelling and analysing biochemical pathways using Petri nets. In *Proc. CMSB*, pages 200–216. LNCS/LNBI 4695, Springer, 2007.

[6] D. Gilbert, M. Heiner, F. Liu, and N. Saunders. Colouring Space - A Coloured Framework for Spatial Modelling in Systems Biology. In J. Colom and J. Desel, editors, *Proc. PETRI NETS 2013*, volume 7927 of *LNCS*, pages 230–249. Springer, June 2013.

[7] M. Heiner, R. Donaldson, and D. Gilbert. *Petri Nets for Systems Biology*, chapter 3, pages 61–97. Jones & Bartlett Learning, LCC, 2010.

[8] M. Heiner, D. Gilbert, and R. Donaldson. *Petri Nets for Systems and Synthetic Biology*, volume 5016 of *LNCS*, pages 215–264. Springer, 2008.

[9] M. Heiner, R. Richter, and M. Schwarick. Snoopy - a tool to design and animate/simulate graph-based formalisms. In *Proc. International Workshop on Petri Nets Tools and APplications (PNTAP 2008, associated to SIMUTools 2008)*. ACM digital library, 2008.

[10] M. Herajy and M. Heiner. Snoopy Computational Steering Framework User Manual Version 1.0. Technical Report 02-13, Brandenburg University of Technology Cottbus, Department of Computer Science, July 2013.

[11] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, and H. K. et al. The Systems Biology Markup Language (SBML): A Medium for Representation and Exchange of Biochemical Network Models. *J. Bioinformatics*, 19:524–531, 2003.

[12] K. hyun Cho, S. young Shin, H. woo Kim, O. Wolkenhauer, B. Mcferran, and W. Kolch. *Mathematical modeling of the influence of RKIP on the ERK signaling pathway*, pages 127–141. Springer-Verlag, 2003.

[13] A. Levchenko, J. Bruck, and P. Sternberg. Scaffold proteins may biphasically affect the levels of mitogen-activated protein kinase signaling and reduce its threshold properties. *Proc. Natl. Acad. Sci. USA*, 97(11):5818–23, 2000.

[14] F. Liu and M. Heiner. *Petri Nets for Modeling and Analyzing Biochemical Reaction Networks*, chapter 9, pages 245–272. Springer, 2014.

[15] W. Marwan, C. Rohr, and M. Heiner. *Petri nets in Snoopy: A unifying framework for the graphical display, computational modelling, and simulation of bacterial regulatory networks*, volume 804 of *Methods in Molecular Biology*, chapter 21, pages 409–437. Humana Press, 2012.

[16] L. Napione, D. Manini, F. Cordero, A. Horváth, A. Picco, M. D. Pierro, S. Pavan, M. Sereno, A. Veglio, F. Bussolino, and G. Balbo. On the use of stochastic petri nets in the analysis of signal transduction pathways for angiogenesis process. In *CMSB*, pages 281–295, 2009.

[17] K. R. Sanft, S. Wu, M. K. Roh, J. Fu, R. K. Lim, and L. R. Petzold. Stochkit2: software for discrete stochastic simulation of biochemical systems with events. *Bioinformatics*, 27(17):2457–2458, 2011.

[18] M. Schwarick. *Manual: Marcie - An analysis tool for Generalized Stochastic Petri nets*. BTU Cottbus, Dep. of CS, 2011.

[19] A. Sinha. Comparison of Stochastic Simulation Tools. Technical report, Brandenburg University of Technology, Computer Science Dep., June 2014.

Appendices

A Accuracy

A.1 Correctness of exports.

In order to perform an export we must be sure about its correctness. We have incorporated two exports in our report. So how do we know that our exports are actually right or wrong. This is a question which could and always will be raised. So, here's a proof for correctness.

Export for Stochastic Petri Nets. In this export we exported the stochastic Petri nets to SBML level 1 and, not the level 2 (we already had the export to SBML level 2). So we were only concerned about the correctness of SBML level 1 file. There's a tool on the market called Dizzy, a stochastic simulation tool, which has a feature of importing SBML Level 1 file. We already know the results that the Snoopy will produce if we simulate our given Petri net. So what we did, we performed our export to Level 1 using Snoopy, and then we imported those files in Dizzy. If Dizzy could simulate those files (SBML leve 1)and could produce the same results and plots (as Snoopy), then we could be damn sure that our export for stochastic Petri nets is correct. And guess what! They did match, and we got the same results as expected. So our export for stochastic Petri nets was correct.

So now, lets move to our second export.

Export for Coloured Petri Nets. For this what we did was, we first performed our export from coloured net to SBML (either level 1 or 2) directly. Then in second case, we performed the export of coloured net to stochastic nets and from stochastic Petri nets to SBML i.e. in two steps. Then, we comapared both the SBML files. We found them to be exactly the same. By same, I mean the exact carbon copy of each other. So this proves that our export for coloured Petri nets are correct.

So I guess now, no one in the world will ever raise the question on the correctness issues of my exports.

B How to reproduce the results ?

In this section we will talk and discuss in detail what one needs to know in order to reproduce the results reported. First thing first, use a system which has the same configuration which I used. Do not jump onto any system, otherwise results may vary. Refer Section 5 for system configuration details.

B.1 BioNetGen

- Inorder, to replicate the results of BioNetGen you should first have bngl file with you. There are two ways of getting that bngl file, first, you can write your own bngl file or second, you can produce bngl file from SBML file(.xml). We used the second approach.

- We used an experimental SBML-to-BNGL translator that you can try at `http://ratomizer.appspot.com/translate`. Note that SBML is a "flat" language, i.e., the molecules don't have internal structure. A basic SBML-to-BNGL translation will therefore give you a flat model. There is a feature in this translator, however, that attempts to infer structure out of an SBML model. You can try this by clicking the "Atomize" box . Try it both ways and see which one works best for you. However, we did not use the atomize option.

- After you have bngl file with you. You can start your simulation. But, I don't know why when I started the simulation I got some errors. Therefore, I did some editing in the original bngl file. So open your bngl file with any texteditor and make these changes. The changes done by me in the original bngl file was -

  - Compartment section in the .bngl file was removed.
  - In reaction rule section, rate constants for all reaction were changed from none to appropiate values.
  - Then those rate-constants were added up in the parameter section.
  - And, the string "@compartment:" was removed from the seed species section.

- Now, inorder to perform simulation you need to write certain commands in action section of bngl file. But, since we are dealing with multiple simulation runs you will find that we do not have any specfic command for it.

Note -Unfortunately, there's not a simple argument that you can pass for running multiple SSA simulations. What many people do is write a shell script (bash, python, etc.) to call BioNetGen on a model file many times. This has a number of drawbacks, however, including having to call the expensive generate_network command over and over. Probably the best approach is to append your model file with multiple 'simulate' commands, each followed by a 'resetConcentrations' command, i.e.,

```
generate_network({overwrite=>1})
simulate_ssa({suffix=>"ssa",t_end=>100,n_steps=>1000})
resetConcentrations()
simulate_ssa({suffix=>"ssa",t_end=>100,n_steps=>1000})
resetConcentrations()
...etc.,
```

You can have as many of these as you like, there's no limit.

There's also a way to do this in BioNetGen using the 'parameter_scan' action. 'parameter_scan' takes all of the same arguments that the 'simulate' action does, plus four additional: *parameter*, *par_min*,*par_max*, and *n_scan_pts*. If you set *par_min* = *par_max* then BioNetGen will run the same system as many times as *n_scan_pts*. For example, you can run 100 SSA simulations using,

```
generate_network({overwrite=>1})
parameter_scan({method=>"ssa",t_end=>100,n_steps=>1000,parameter=>"k",
par_min=>1,par_max=>1,n_scan_pts=>100})
```

Here, k is just a dummy parameter. You could add it to your model or you could just specify a parameter that already exists in your model and set par_min and par_max equal to the value of that parameter.

- After making the appropiate changes in bngl file and writing the specific actions which you want to perform. Now you have the real bngl file with you. Save that bngl file with appropaite name.
- Open the terminal. And move into the directory where you have saved your bngl file (model file). Then write the following command.

```
perl <BNGroot>/BNG2.pl <modelfile>.bngl
```

BNGroot is the complete path where you have saved your BioNetGen package.

- If everything goes well you'll get your output. Note the simulation time.

  Now the catch is, if 100 runs are performed the tool will output 100 result files, and not the average of those 100 files. So this requires a large disk space if simulation run is a million times, because it will output a million files.However for plotting the graphs (not the comparision one) we have considered the average of those 100 files (in case if runs=100). And for taking the average of those 100 files a JAVA code was written.

- After the simulation, its time for computing the average. For averaging as told earlier, a JAVA code was written.

- Now we can proceed for plotting the cuve. This can also be done in two ways.

  - First way is, when you do the averaging of files you can save your averaged file in .cdat format. then, you can use the "PhiBPlot" which comes with BioNetGen package. PhiBPlot is basically a jar file. You can open your .cdat file with it and look the curves.
  - Or the second way is, you can save your averaged file in .txt format. And then you can use the "gnuplot" for plotting the curves.

  I used both ways. But, it doesnot matter, you can use either of the two ways. Save your plots if required.

B.2 SSC

- For SSC one needs a rxn and cfg file for simulation. This can be generated in two ways -

  - Either you can write your own rxn file. Or ,
  - You can generate rxn file from bngl file. BioNetGen to SSC translator provides a SSC equivalent model to your BNG model. The translator outputs two files, one with the translated rules, as modelName.rxn, and the other containing the definition of variables in them, as modelName.cfg.The commands used to generate .rxn and .cfg files are "writeSSC()" and "writeSSCcfg()" respectively.These commands are written in the action block of .bngl file. We used this second method.

- After writing those commands. Start the simualtion of bngl file. After successful completion of bngl file you could see that two more files are generated namely, rxn and cfg file.

- Now change the floating values of "Initial molecules and their concentrations" in rxn file to integer values. After this you are good to perform simulation with SSC.

- Simulation of rxn files

  Simulating models written in SSC consists of two steps: compiling the model, which expands the pattern-based description into all possible species and reactions, and running the resulting simulator executable, which actually carries out the simulation.

  - Compiling and debugging

    Once the model has been written to a file (say, model.rxn), it's compiled by running *ssc model.rxn*

    which produces output resembling
    reading: model.rxn...
    expanding reactions...
    expansion complete after 2240 steps: 135 compounds and 1120 reactions
    simulator executable: model

    and a simulator executable, in this case called model.

  - Simulating

    The easiest (and fastest) way to run the simulator is to specify the -e flag followed by simulation end time (in seconds). When the simulation finishes, it will output the final time together with the counts recorded by the various record statements, separated by TAB characters. The simulation may finish before the specified end time if no more reactions are possible; this generally does not happen in spatially resolved simulations because, although all reactions may have run out of reactants, diffusion can still take place.

    We can also produce a trajectory sampled at regular intervals by adding the -t flag. When some constants (reaction rates or counts) were specified as variables in the model file, the simulator must be provided with a configuration file containing the variable values with the -c flag.

- So open your terminal and move to the location where you have your rxn file. Then as explained above for compiling write

  ```
  <SSCpath>/ssc modelFile.rxn
  ```

  the following command. This will create an executable.

- And finally for simulation write the following command.

  ```
  ./modelName -T 1 -e 100 -c modelName.cfg -o outputFile.trj
  ```

  So this command will perform only one simulation run. Inorder to perform more than one simulation run a script was written. But, we can use this script for performing one simulation run also. The script automatically computes the time taken in performing simulation, which is displayed in the terminal. Note this time.

  Note -However ,the problem with SSC is that, if we perform simulation more than once then for each run a .trj file will be created. So if we have 100 runs then 100 .trj files are created. Then for plotting the graph we need a file which is the average of those 100 files. Since .trj is some sort of binary file therefore we need to convert those files to readable format so that we can take the average of those 100 files. But, converting a file from .trj to .txt format (using ssc-trj-reader-0.01.jar) requires huge amount of time. So for million runs simulation was performed however the averaging was not done.

- Now, the next step is to convert trj (trajectory) file to txt file. If we have one trj file then we can directly plot it using SSC 3D Viewer(this can be downloaded by clicking start direcly link under the SSC 3D viewer section on ths page `http://web.mit.edu/irc/ssc/`). However for 100 files (in case if runs=100) we need ssc-trj-reader-0.01 (a jar file) which allows converting SSC-generated trj files to Matlab-readable format.

  We created one script which calls this jar file again and again and produces txt file.

- Now when we have 100 txt file (in case runs=100) then we can do the averaging in the same way as we did for BioNetGen.

- After you have averaged file. Plot the graph by using gnuplot.

B.3 Cain

- Open the tool in the terminal along with memory usage script. e.g.

  ```
  /home/chiru/Desktop/Repressilator/memusg python Cain.py
  ```

  The memory script memusg calculates the peak memory usage by any application.

- Open the coloured petri net. By selecting File->open from the menu bar. And then, selecting the appropiate cain_file.xml

- Select the appropiate method in method editor (i.e. Time homogenous, Time series uniform, Direct, 2-D search) .Also make sure you have appropiate start and end time.

- Launch the simulation by clicking on the "launch action solver by mass action button".This is termed as one experiment. Perform 10 trials of each experiment. Note the simulation run-time displayed by the tool on paper, export the traces.

  Note-However we limited ourselves to 3 trails, due to lack of time.

- Close the tool and record its peak memory usage.

- If simulation runtime $>$ 3,600 seconds. Terminate the simulation.

- A spreadsheet is created manually and the memory consumption and simulation runtime are entered manually. The average runtime and peak memory comsumption can be calculated using the functions available in the spreadsheet.

  Note-It may also be possible that Cain may quit unexpectedly, or it may kill the process or it may give segmentation error. If it does this for more than twice, continously, we will term this as "Cain crash". This can happen while doing simulation or when you are exporting the data. One more thing which I would like to state is that, Cain takes huge amount of time in exporting data. This is just for the record. Even if exporting takes more than 3600 sec we have to continue with the simulation, because the constraint is on the simulation time and not in the exporting time.

B.4 Snoopy

- Open the tool in the terminal along with memory usage script. e.g.

  ```
  /home/chiru/Desktop/benchmarks/memusg ./snoopy2.sh.
  ```

  The memory script memusg calculates the peak memory usage by any application.

- Open the coloured petri net. By selecting File->open from the menu bar.

- Then select the appropiate "constant" value for the benchmark under the declaration section in Snoopy.

- Start the simulation, by clicking View->Start Simulation-Mode in the menu bar.

- Since it is a coloured Petri net a a dialog box will appear which will ask you to unfold the given net. Select thread count = 8 and then click the start button. If the net is very large, then it may take some time to unfold.

- Go to Current view->edit of the recently opened window. Move all the elements from the "Overall place" to the "Selected Place" by clicking >> this button. Then, hit save button.

- Perform simulation on a particular benchmark for a particular value of scaling parameter, thread and run. This is termed as one experiment. Perform 10 trials of each experiment. Note the simulation run-time displayed by the tool on paper, export the traces.

- Close the tool and record its peak memory usage.

- If simulation runtime > 3,600 seconds. Terminate the simulation.

- Perform 10 trials for each experiment.

- A spreadsheet is created manually and the memory consumption and simulation runtime are entered manually. The average runtime and peak memory comsumption can be calculated using the functions available in the spreadsheet.

B.5 Marcie and StochKit

- Shell script for benchmark is created. The benchmark shell script stops the simulation once the simulation runtime is $>$ 3,600 seconds. The shell script stores the output of the terminal for a particular experiment in a .out file. This shell script calls the memory usage script in order to compute the peak memory consumption. A .csv is created where the memory consumption and .out is creates where the runtime of the tool is written.

- Once the .csv file containing the memory consumption and .out file containing the runtime is created, we parse all these files and note down all these data in spreadsheet.

- Shell script for StochKit and Marcie is created which calls the benchmark shell script along with the command line syntax for Marcie and StochKit.

  Command line syntax for:

  Marcie

  ```
  <marcie_path> --simulative --net-file= <net_file_path>
  --sim-stop= <sim_stop_time> --sim-out-steps= <no_of_interval_steps>
  --const <value_of_scalableParameter> --threads= <value_of_thread>
  --sim-result-file= <output_file_path/output_file_name>
  ```

  Note: The net file provided to Marcie is in apnn format. For more information on using marcie commands please refer [18].

  StochKit

  ```
  <stochkit_driver_name> -m <model_name> -t <end_time_interval>
   -r <no_of_runs> -i <interval_ step_count> -p <thread_value> --label
   --out-dir <output_file_path/output_file_name>
  ```

  For more information about the StochKit commands please refer the user manual for StochKit.

C Everything you need to know about Plots.

C.1 Runtime, Memory Consumption and Disk Consumption Plots

- Runtime graphs are plotted for a specific value of thread and specific value of runs. The x axis denotes the scaling parameter and the y axis denotes the simulation runtime (in sec).
- For graphs which donot have a specific value of thread (like the BioNetGen and SSC), they are plotted for a specific value runs. The x axis denotes the scaling parameter and the y axis denotes the simulation runtime (in sec).
- Memory consumption graphs are plotted for a specific value of thread and specific value of runs. The x axis denotes the scaling parameter and the y axis denotes the memory consumption (in KB).
- Disk Consumption graphs which donot have a specific value of thread (like the BioNetGen and SSC), they are plotted for a specific value of runs. The x axis denotes the scaling parameter and the y axis denotes the disk consumption (in KB).
- You may find certain graphs which contains less number of points than the others. for example, consider the *Figure 26*, you can clearly see that for thread value =8 and runs =10000 the cain contains only 3 points, whereas its companion tool contains more points. This is because, here cain had less datapoints to be plotted as it lost its data points because of Cain crash.
- The graphs are log scaled.
- Graphs are plotted using gnuplot.
- We have some figures which donot contain the plots for run = 10000, this is because they did not had enough datapoints with them. Some data points represent either the tool crash and the others represent that they take took more than 3600 sec for simulation. Hence we are left with very few datapoints. And, hence no plots for them.