Meta-stochastic Simulation of Biochemical Models for Systems and Synthetic Biology

Daven Sanassy, Paweł Widera, and Natalio Krasnogor*

Interdisciplinary Computing and Complex BioSystems (ICOS) research group, School of Computing Science, Newcastle University, UK

E-mail: natalio.krasnogor@newcastle.ac.uk

Abstract

Stochastic simulation algorithms (SSAs) are used to trace realistic trajectories of biochemical systems at low species concentrations. As the complexity of modelled bio-systems increases, it is important to select the best performing SSA. Numerous improvements to SSAs have been introduced but they each only tend to apply to a certain class of models. This makes it difficult for a systems or synthetic biologist to decide which algorithm to employ when confronted with a new model that requires simulation. In this paper, we demonstrate that it is possible to determine which algorithm is best suited to simulate a particular model, and that this can be predicted *a priori* to algorithm execution. We present a web based tool *ssapredict* that allows scientists to upload a biochemical model and obtain a prediction of the best performing SSA. Furthermore, *ssapredict* gives the user the option to download our high performance simulator *ngss* preconfigured to perform the simulation of the queried biochemical model with the predicted fastest algorithm as the simulation engine. The *ssapredict* web application is available at `http://ssapredict.ico2s.org`. It is free software and its source code is distributed under the terms of the GNU Affero General Public Licence.

*To whom correspondence should be addressed

Keywords

stochastic simulation, web application, model properties, algorithm performance, systems biology, synthetic biology

1 Introduction

Simulation of mathematical and computational models of reaction networks is an invaluable tool for scientists aiming to understand the dynamic behaviour of complex biochemical systems. In the fields of Systems and Synthetic Biology, repeated rounds of model-driven hypothesis generation, validated or refuted by wet lab experimentation, lead to refined quantitative and predictive models, and ultimately better bio-designs. *In silico* experimentation with these models is cheaper, faster, and more reproducible than its real world counterpart (i.e. wet lab). In order to maximise the potential of *in silico* experimentation, it is important to ensure that (1) simulation algorithms can scale efficiently with the class and size of problem and (2) optimised reference code is readily available, well documented and easy to reuse.

Stochastic Simulation Algorithms (SSAs) are the primary means of simulating naturally discrete cellular systems affected by stochastic noise. They generate multiple *realistic* trajectories of molecular quantities over time given an initial state (e.g. species counts), a set of reactions (with associated stochastic rate constants) and stopping criteria. SSAs are an important

tool in systems biology software suites such as *Infobiotics Workbench* (*1*) and have been used for theoretical work (*2–4*).

Exact SSAs, introduced by Gillespie (*5*), generate trajectories that are demonstrably equivalent to the *Chemical Master Equation* and must simulate each and every reaction in the system. SSA algorithmic complexity being $O(M)$ (where M is the set of reactions), and concomitant generation of pseudo-random numbers to emulate stochasticity for each reaction event, makes simulating ever larger reaction networks increasingly intractable despite continued advances in computational power. This has led to many studies addressing point (1) above, namely how to improve the scalability of the SSA. For example, *approximate* SSAs have been introduced that conditionally apply multiple reactions at each step (*6*) as well as optimised *exact* variants that use improved data structures to accelerate computations (*7–12*).

However, the availability of multiple variants of the SSA comes at the cost of a lack of clarity as to which one to use for a particular biochemical model. More specifically, many published SSAs are tested with an insufficient number of models, mostly tailored to properties of the newly introduced algorithm. Consequently, it is hard to extrapolate or compare performance between algorithms as each will often be benchmarked against competitors' algorithms using only biochemical models that perform favourably with the newly introduced algorithm. When considering these variants, a scientist may wish to know which SSA will be the fastest for simulating their particular model(s).

Currently, it is common to execute reaction networks with a single SSA implementation, for example Next Reaction Method (NRM) (*7*). Due to the lack of model and algorithmic analysis available, scientists are unaware that a different algorithm may perform orders of magnitude faster than their "default" algorithm. To compound this issue, whilst several stochastic simulators are freely available (*13–16*), their selection of algorithms is limited. Such a situation would result in a scientist limiting the complexity of their model to obtain a tractable simulation time. Therefore, it is preferable that scientists are provided with tools that match the best algorithm for their model and allow for better performing simulations. If simulation time can remain tractable in spite of increasing model complexity, the development of finer grained biological knowledge is possible.

The cost of simulating a system with one SSA variant or another depends on the properties of the underlying network of the model and the states reached during the simulation. Each biological model exhibits characteristics that may be suited to a particular simulation algorithm. Thus effective discrimination between SSAs should be based on matching model characteristics to algorithmic performance. Network analysis is an important aspect of Systems and Synthetic Biology, but Roy (*17*) notes that usually only a few "handpicked" network properties are considered to determine the influence of the network structure. As an example, the *StochKit2* (*13*) simulation software selects algorithm based on model "properties" but actually only considers the number of reactions in the model and is therefore very limited in discrimination. A prediction based on a comprehensive evaluation of network properties is required to discriminate between the significant number of SSA variants that exist as a function of their performance with specific biochemical models.

To address these problems, we have created a complete meta-stochastic simulation solution implemented as a user-friendly web application called *ssapredict*. Our tool allows a scientist to upload their model and automatically predicts the best algorithm to use. Once a prediction has been made, the user can download *ngss*, our portable high performance simulator, preconfigured to run their model with the predicted fastest algorithm. Our web application tackles three issues: (a) it simplifies the decision making process for the scientist, (b) it produces an acceptably accurate prediction of SSA-to-model match, and (c) it standardises SSA source code contributing to reproducibility of *in silico* experiments.

The *ngss* simulator and *ssapredict* web application are under continuous development and we are open to integrating new algorithmic

methods as well as to extend the set of models we use for training. In the long term, we would like to lead an effort, in collaboration with other scientists in the field, to construct an open benchmark composed of both algorithms and models. This will not only result in improved prediction accuracy but also increased simulation speed for a wide range of models.

2 Results and Discussion

2.1 Stochastic Simulation

Our experimental analysis included eight *exact* SSA formulations that we had implemented. These are Direct Method (DM) (*18*) and First Reaction Method (FRM) (*5*), Next Reaction Method (NRM) (*7*), Optimised Direct Method (ODM) (*10*), Sorting Direct Method (SDM) (*8*), Logarithmic Direct Method (LDM) (*9*), Partial Propensities Direct Method (PDM) (*12*) and Composition Rejection (CR) (*11*). An *approximate* algorithm, Tau Leaping (TL) (*6*) was also implemented and analysed. A brief description of these algorithms is provided in Section 3.1.

In our experiments, we used models taken from the *BioModels* database (*19*) and the performance (reactions executed per second of CPU time) of each of the algorithms was measured for every model. The distribution of best performing algorithms across all models is shown in Table 1.

Table 1: Distribution of best performing algorithms for all 380 models from the BioModels dataset. The first row shows how many times a particular algorithm was the fastest for a model in the dataset. The same value is also displayed as a percentage of the total (second row).

CR	DM	NRM	FRM	LDM	SDM	TL	ODM	PDM
0	1	1	2	9	43	75	87	162
0.00%	0.26%	0.26%	0.53%	2.37%	11.32%	19.74%	22.9%	42.63%

To put this result into perspective, we compared the performance of each algorithm to the best algorithm in the group for each of the models. Figure 1 shows that three frequent winners PDM, ODM and SDM, have very similar performance profiles (they are all improved variants of DM) with the notable exception of a few models for which PDM performs badly. TL, another algorithm in the top 4, performs exceptionally well for about 20% of the models, but is outperformed by other algorithms for the rest of the dataset. For the worst performers, CR and FRM, there is a clearly visible gap that separates their performance from the best.

In Table 2, we show how consistent the top 4 algorithms are. For each algorithm in the top 4, we measure how many times it was ranked below the top 4. ODM was the algorithm that most consistently remained in the top 4 (378 out of 380 models), but was closely followed by SDM (368 out of 380 models). On the other hand, PDM and TL were ranked below the top 4 many times, including being ranked as the worst algorithm for some models. TL in particular remained in the top 4 for only 80 models.

Table 2: Number of times one of the 4 best performing algorithms (See Table 1) was ranked below the top 4 for any of the 380 models from the BioModels dataset. Each row shows the total number of models for which an algorithm was ranked under a given threshold. The lowest possible rank was 9.

rank	ODM	SDM	PDM	TL
> 4	2	12	42	300
> 5	0	1	22	287
> 6	0	0	17	205
> 7	0	0	7	51
$= 9$	0	0	6	8

2.2 Performance Prediction

We define meta-stochastic simulation as an automatic methodology for selecting the best algorithm for a given model. Different SSAs have varying performance profiles dependent on a particular model's properties. Stochastic models can be represented as graphs of dependencies between reactions or species. Using these graphs we are able to build a topological profile

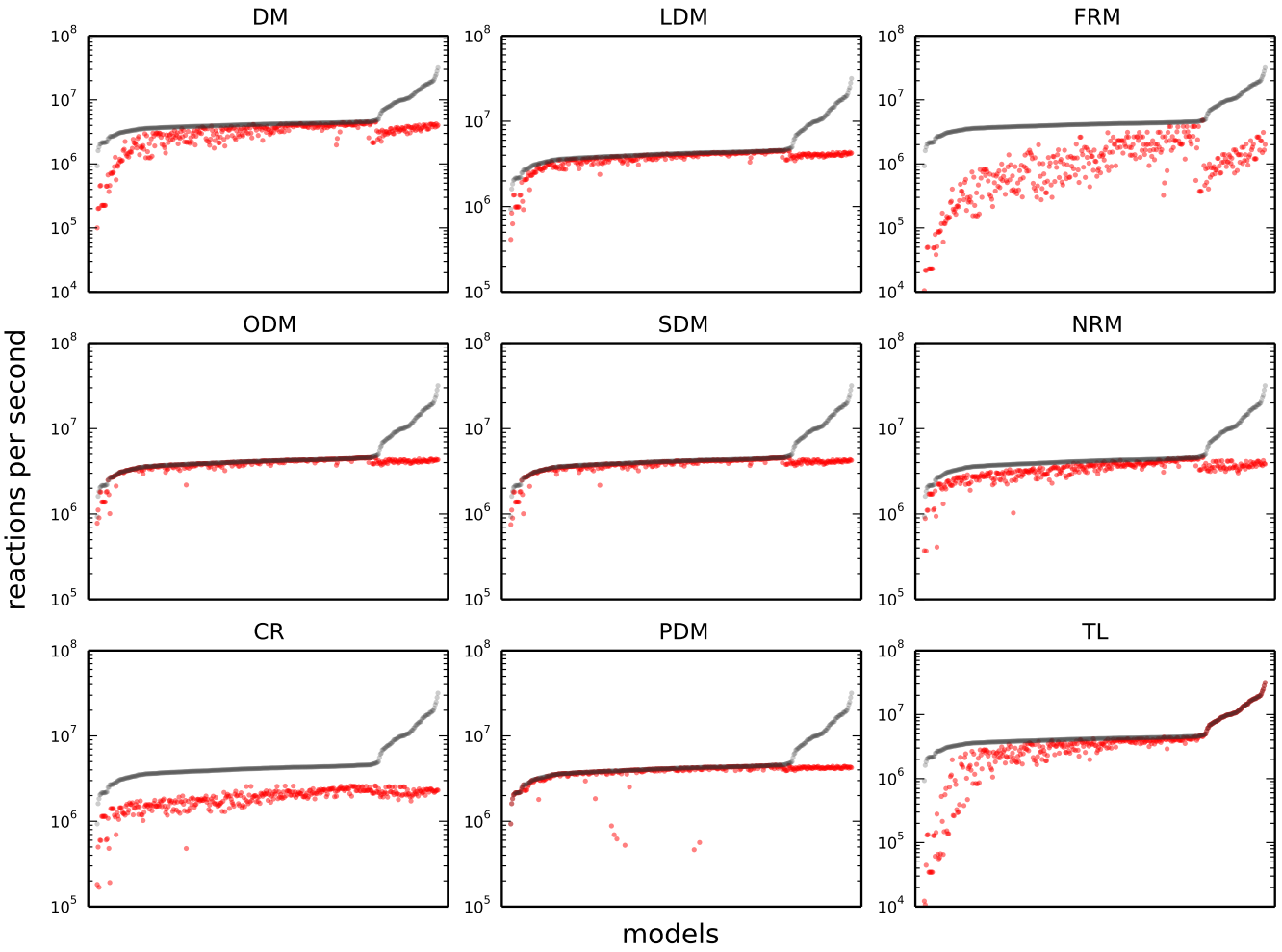

Figure 1: Comparison of the performance of each algorithm against the best performance for each model. The red data points are the algorithm performance values for each model. The grey data points show the best performance for each model. The models on the horizontal axis are ordered by best performance.

of each model. We use the topological properties to learn how to predict the fastest SSA (from a pool of nine algorithms) for a given model.

An example of what we include in the model topological profile is the number of nodes or the number of connections in the reaction or species dependency graph. Other properties contain information about the connections of individual nodes (node degree), the entire graph (graph density) or the existence of mutual dependencies (reciprocity). All topological properties used in our analysis are discussed in Section 3.3 and listed in Table 5.

To provide the reader with an intuition of which graph topological properties might be useful in algorithm performance estimation, we performed a small case study. For two top 4 algorithms, PDM and TL, we selected 5 models for which their performance is the worst and 5 models for which it is the best. We then measured a median value of each topological property for the two sets (*worst* and *best*) of models. Finally, we selected 10 properties for which the absolute difference between *median(worst)* and *median(best)* is the highest. We found that the worst models for PDM had high reaction graph density and total degree, high species graph min degree and medium species reciprocity. Interestingly for TL, the worst models had low reaction graph density and low species graph min degree, which is the opposite of the worst models for PDM.

In Figure 2, we show the distribution of nor-

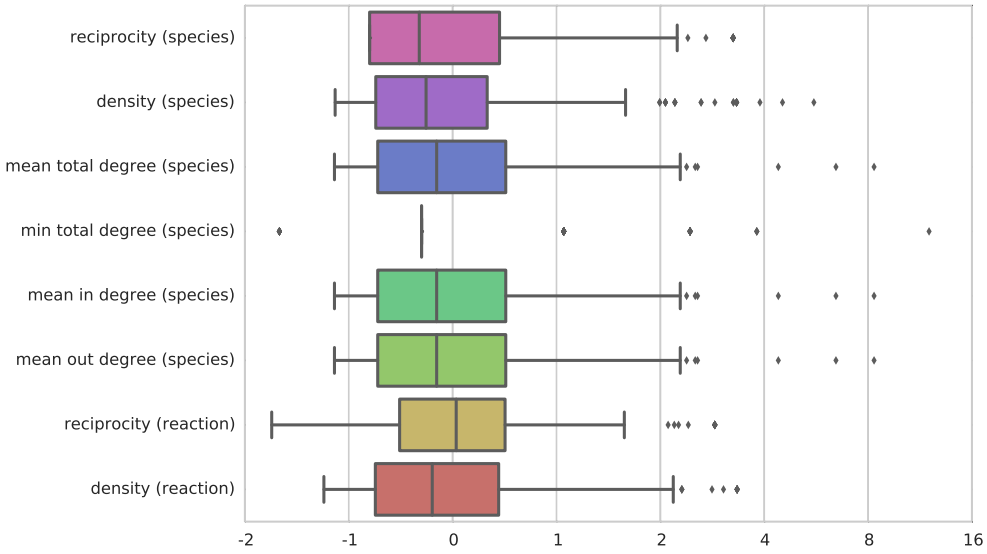

Figure 2: Distribution of values of selected topological properties. For all properties, the mean value is equal to 0 and the standard deviation is equal to 1. The whiskers represent the 1.5 IQR (interquartile range) past the closest quartile (left/right edge of the box). Observations outside this range are marked as outliers. Horizontal axis is on a logarithmic scale.

malised values of properties selected for both algorithms. The species graph min degree stands out as having a dominating value (here 1) with many outliers. For other properties the range of values is much broader but outliers remain frequent, indicating possible hard to predict edge cases.

2.2.1 Performance Estimation

Linear regression (ordinary least squares method) was used to fit a linear model estimating the performance. Regression was performed on a per algorithm basis with results shown in Table 3. The coefficient of determination (R^2) is close to 1 (perfect fit) for 7 out of 9 algorithms. PDM and TL algorithms are the exception with R^2 of 0.71 and 0.6 respectively, which indicates that their performance is more difficult to estimate with a linear function.

Table 3: The linear regression fit of algorithm performance on all models measured with the coefficient of determination (R^2).

TL	PDM	NRM	CR	FRM	SDM	ODM	DM	LDM
0.600	0.712	0.971	0.973	0.979	0.985	0.986	0.989	0.991

2.2.2 Prediction of the Fastest SSA

Employing 10-fold cross-validation, we aimed to determine the quality of algorithmic performance predictors that could be generated from the BioModels dataset using model topological properties.

Some of the analysed model properties are impractical for use in a prediction tool. This is because with large reaction networks those properties can require even more computational time to compute than the simulation time of the model. A topological features set that was *fast* to compute (as these would be the strongest candidates to be used in a tool that could predict algorithm performance of a particular model *a priori* to simulation) was therefore identified (see Table 5 in Section 3.3).

We used two variants of a random predictor and four predictors based on well known methods (linear regression, logistic regression, support vector classifier and a nearest neighbour classifier). For each predictor we performed a 10-fold cross-validation experiment and measured the mean accuracy and standard deviation of the predictions. The accuracy of each predictor is tested with four tolerance thresholds ($\varepsilon = [0\%, 1\%, 5\%, 10\%]$). In this scenario,

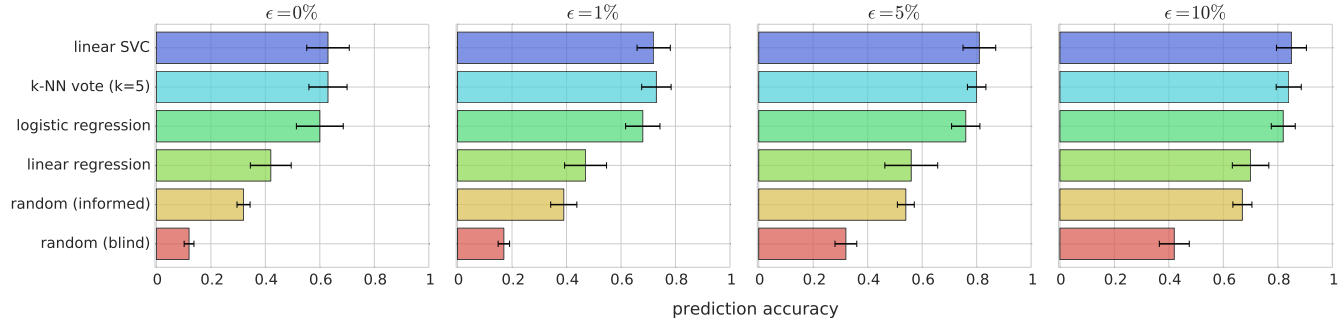

Figure 3: Results of the 10-fold cross-validation classification experiment using a reduced set of properties (computational complexity $\leq O(V + E)$) with increasing tolerance threshold ε.

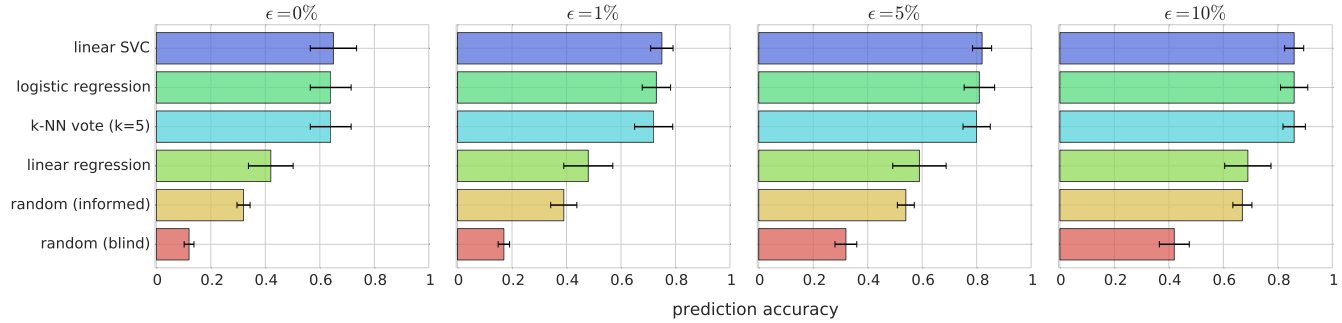

Figure 4: Results of the 10-fold cross-validation classification experiment using the total set of available properties with increasing tolerance threshold ε.

if the performance of a best algorithm selected by a predictor for a given model lies within the tolerance threshold of the performance of the actual best algorithm, the prediction is scored as correct because the performance of the predicted best SSA only differed from the actual best SSA performance by an acceptable amount (e.g. 10%).

For the first experiment, we decided to assess the performance of predictors when given the full set of 32 computationally inexpensive (*fast*) properties. Results displayed in Figure 3 demonstrate that all classifiers perform better than a random selection. k-Nearest Neighbour and linear SVC had the same prediction accuracy (63% with $\varepsilon = 0\%$), which is a significant improvement over a blind random selection (12% with $\varepsilon = 0\%$). However, from evaluating the prediction quality at differing tolerance thresholds, it appears that linear SVC was the strongest predictor (85% with $\varepsilon = 10\%$). For comparison, a blind random selection at $\varepsilon = 10\%$ results in half the accuracy at 42%.

For a comparative analysis, we subsequently tested the prediction accuracy with the entire set of 100 slow and fast topological features, results are shown in Figure 4. Prediction accuracy overall was similar to the previous (fast properties) experiment but demonstrated some slight improvement. Linear SVC was still the strongest predictor and had improved from 63% to 65% (85% to 86% with $\varepsilon = 10\%$). As expected, higher quality predictions occur when more properties are introduced in the cross-validation experiments (effectively testing on the training set). However, it is important to note that using just the computationally inexpensive properties produced results of similar quality to the full set of properties. Therefore, the strong performance of the predictors with fast properties means that we can create a tool that produces an accurate prediction quickly.

2.2.3 Prediction Inaccuracies

Often the failures of a tool are as important as its successes. To demonstrate the consequences of mispredictions of *ssapredict*, we ran another

10-fold cross-validation experiment. This time we measured the related relative performance loss for each inaccurate prediction. Figure 5 shows the distribution of these values for the best predictor in the group (linear SVC) compared to the random predictors. The median value for the best predictor was only 5.4% and the interquartile range was lower than 20%. This means that most of the mispredictions correspond to less than a 20% performance loss. However, we also found several outliers for which the performance drop was large (up to 86%).

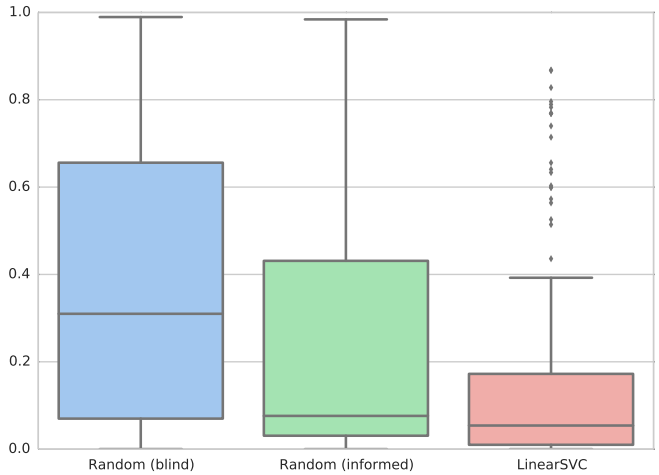

Figure 5: Distribution of relative performance loss caused by mispredictions for the best (linear SVC) and worst predictors (two variants of random choice). The whiskers represent the 1.5 IQR (interquartile range) past the closest quartile (top/bottom edge of the box). Observations outside this range are marked as outliers.

2.2.4 Large Scale Model Experiments

In our final experiment, we investigated whether the predictors trained on small models would be accurate for large models. Figure 6 shows the algorithmic performance for the *Escherichia coli* quorum sensing model instantiated on 1×1, 10×10 and 100×100 two-dimensional lattices (see Table 4 in Section 3.2 for the model network sizes).

The linear SVC predictor selected TL for the single cell lattice and PDM for the 10×10 and 100×100 lattices. Whilst the TL prediction is accurate, for the large lattices CR is the fastest algorithm. The PDM algorithm is 13% slower than CR for the 10×10 lattice and 82% slower for 100×100.

2.3 Web Application

Our meta-stochastic simulation solution, *ssapredict*, is implemented as a web application. *Ssapredict* is designed to be easy to use and receiving a prediction only requires the user to press an upload button and select the biochemical model of interest (in SBML (*20*) format) that resides on their computer. Once the prediction has been made, the user can download our portable high performance simulator, *ngss* (next generation stochastic simulator) which is preconfigured to run the model with the predicted fastest algorithm.

Figure 7 shows a screen shot of the results after *ssapredict* model analysis. Model properties are displayed along with the predicted fastest algorithm. The *simulate model* button allows the user to download the preconfigured *ngss* binary for their platform.

2.4 Discussion

Our proof-of-concept SSA performance predictor has demonstrated that even using a limited set of topological properties it is possible to pick a fast algorithm for a given model. The prediction accuracy was found to be much higher than a random choice. In addition, this work has highlighted that no single SSA is superior to all others for all models. For example, for models with high min degree in species graph TL outperforms PDM, but when that degree is low, PDM outperforms TL.

Looking back at the distribution of fastest algorithms (Table 1 and Figure 1), some results are to be expected. For example, FRM being a weak performer, and a more sophisticated algorithm such as PDM being a strong performer. However, we see some unexpected results too, such as one of the most sophisticated algorithms, CR, *never* winning and NRM being a less frequent winner than FRM. Table 2 demonstrates that whilst an algorithm may be

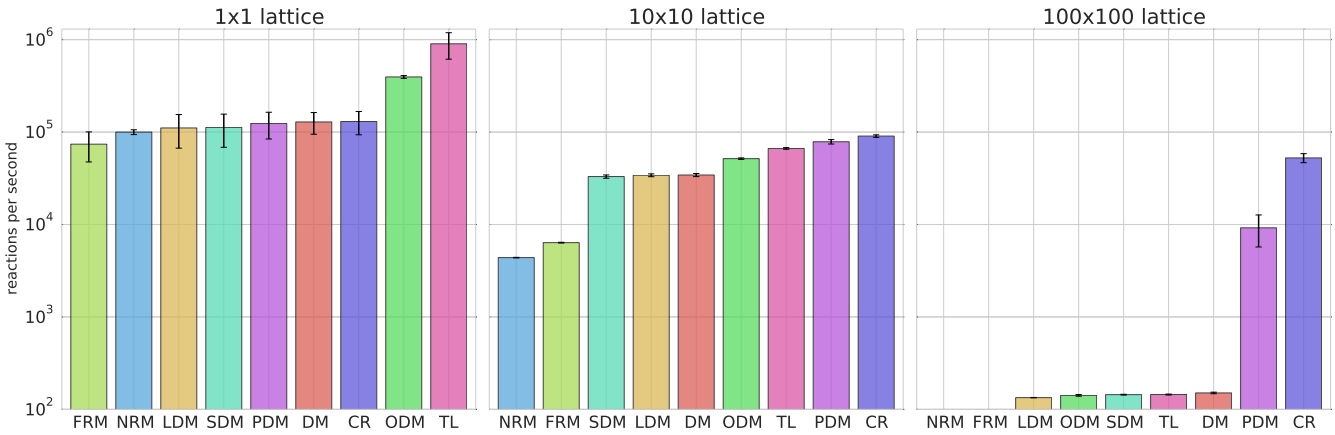

Figure 6: Average performance for the *Escherichia coli* quorum sensing circuit. The model was instantiated on a square lattice with 1, 100 and 10 000 points. The algorithms on the horizontal axis are sorted from the slowest to the fastest and uniquely coloured. The vertical axis shows algorithmic performance in reactions per second and is on a logarithmic scale. The error bar shows the standard deviation across 10 runs.

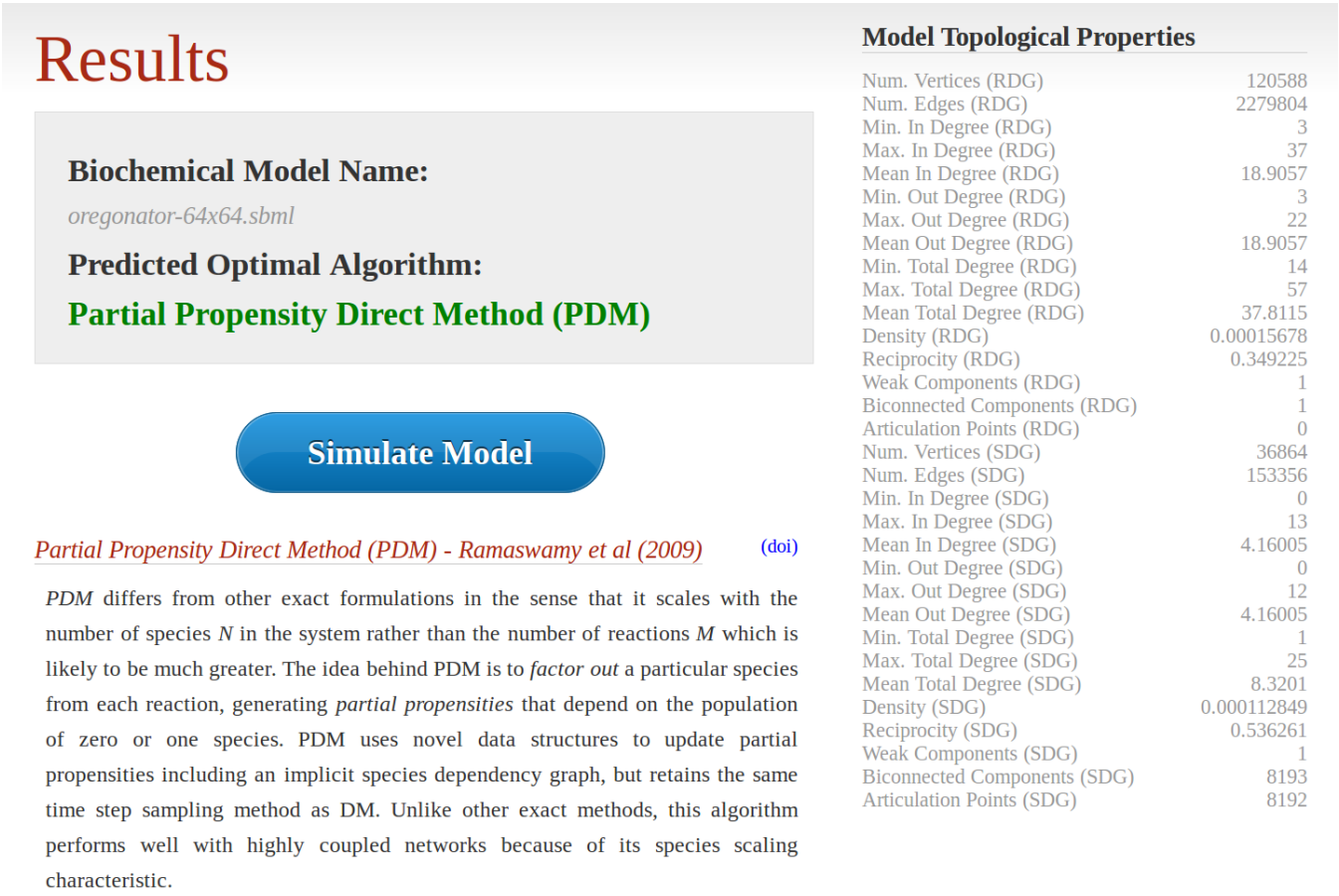

Figure 7: Screenshot displaying the results of analysis performed on a model by *ssapredict*. Model topological properties can be inspected and a prediction of fastest algorithm is displayed. There is an option to simulate the model.

considered the fastest for a significant number of models, there are also models for which it may perform poorly. TL was the fastest algorithm for 75 out of 380 models, but was also ranked below the top 4 algorithms for 300 models. This result highlights the importance of selecting an algorithm on a per model basis rather than using a single algorithm for all models.

The large scale model prediction experiment results (see Figure 6) show that CR is the fastest algorithm for these larger models. Due to lack of models of this scale in our training set, *ssapredict* is unable to predict that result. However, it does select PDM, the second fastest algorithm in the group. This is one of the limitations in predictor accuracy — it is highly dependent on the size and variability of models used for training. In order to train better quality predictors, a larger number of models is needed, preferably containing special instances for which each of the algorithms was designed to perform best with.

While there are many biochemical models available in the literature and online, there are few that are specifically intended for stochastic simulation. Most of the models we used from the BioModels dataset were deterministic models and to simulate them stochastically we fixed rate constants. Whilst there are many complete deterministic models available from online databases, few complete curated stochastic models are freely available. Therefore, a future analysis featuring complete stochastic models will have to be preceded by the creation of a dataset with a large number of curated stochastic models.

Initial species amounts were set to a constant 100 to complement the static topological analysis. A major limitation of our current analysis is that it will not be able to account for transient changes that occur within a simulated stochastic model. This is important because high copy numbers of chemical species can create intractable simulation conditions for *exact* algorithms.

With a larger and more diverse training set, we should be able to minimise the cost of mispredictions. Although currently it is often small (median value of the relative performance drop was 5.4%), there are edge cases where the predictor is choosing an algorithm with 15% of the performance of the best algorithm in the group.

We believe that future work in this area must include a comprehensive benchmark of algorithms with a large set of models that exhibit varied characteristics. This work can be considered a precursor to producing a meta-stochastic simulator that can dynamically switch algorithms when transient changes that affect performance occur in a biochemical system. As Synthetic Biology aims to compose large systems from smaller biological components, it is expected that the size of models will increase rapidly in the future. These large systems will be intractable for simulation without further algorithmic innovation. In our opinion, meta-stochastic simulation is one of the ways to keep up with the growing requirements of Synthetic Biology.

We invite scientists to contribute biological models and stochastic simulation algorithms that we could use to improve *ssapredict*. Submission instructions are available at the *ssapredict* website: `http://ssapredict.ico2s.org`. We intend to periodically re-train the predictor with a growing set of models and to add more algorithms to the *ngss* simulator.

3 Methods

3.1 Simulation Algorithms

In this section we will give a brief review of the algorithms available for prediction in *ssapredict* and consequent simulation in *ngss*. FRM was the first SSA introduced by Gillespie in 1976. FRM is a simple algorithm that requires a random number generated for each reaction in the system every algorithmic iteration (*5*). DM was introduced to replace FRM and only required two random numbers generated per algorithmic iteration. DM is the *de facto* standard SSA formulation and is still commonly used for simulation (*18*).

NRM (based on FRM) used a large number of optimisations including the use of a reaction dependency graph to minimise propensity recal-

culation and a priority queue to find the next reaction to fire. Furthermore, NRM only required one random number generated per iteration (*7*). ODM builds on DM with a reaction dependency graph and reduces reaction search depth by performing a pre-run and sorting reactions by their likelihood of firing (*10*). SDM is similar to ODM but dynamically sorts reactions during algorithm execution. This means that SDM performs well with simulations that experience significant transient fluctuations (*8*). LDM is similar to ODM and SDM but uses a binary search to reduce reaction search depth (*9*).

PDM builds on DM but uses a species dependency graph and factors out reaction propensities by species. PDM claims to be superior for highly coupled reaction networks (*12*). CR is another DM variant which uses rejection sampling to select the next reaction and thus claims constant time scaling. Therefore, in theory, CR should outperform other exact algorithms when presented with large reaction networks. CR uses composition (reaction propensity grouping) to reduce the number of rejections per algorithmic iteration (*11*).

In addition to the exact SSAs, there are many approximate and hybrid algorithms (*6*, *21–23*). At this stage, we focused on exact methods and have only included one approximate method (*explicit* TL). We selected TL, as it is the first hybrid method introduced by Gillespie and can be considered as the *de facto* standard hybrid SSA formulation. TL is an approximate algorithm that applies many reactions per algorithmic step (rather than one per step as in the exact formulations). This is dependent on the propensities of the system being relatively stable, otherwise the algorithm reverts to DM (*6*). We used an updated version of TL that incorporates optimised step size selection (*24*).

As more algorithms are added to the *ngss* simulator in the future, we will also retrain our predictor to offer the best simulation option for the user's model.

3.2 Biochemical Models

We used 380 models retrieved from the *BioModels database* (*19*) in SBML (*20*) format. These models describe various different biological processes and are curated from peer reviewed literature. This gives us access to a large number of models used by scientists with variation in reaction network sizes. In Figure 8, a histogram is shown displaying the spread of model size within the dataset, quantified by the number of reactions in the model. It can be seen from the histogram that the vast majority of BioModels have a reaction network size of 50 reactions or less, but there is also a small number of larger models (up to 1800 reactions).

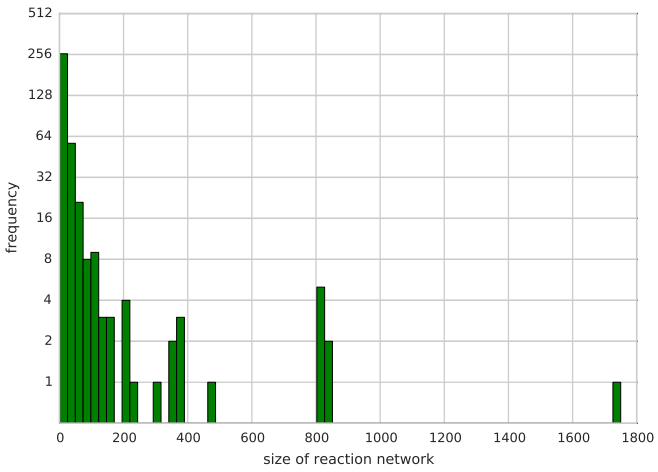

Figure 8: Distribution of the size of model reaction network across the BioModels dataset. The bin size is 25. The vertical axis is on a logarithmic scale.

Our proof-of-concept algorithmic performance predictors only consider static structural model topological properties to test if we could make accurate predictions with that data. Therefore, alterations were made to models to focus on the model structure analysis. In order to simplify models and remove extra variables that cannot be captured by the dependency graph analysis, the amounts of all species were set to 100 and remain constant throughout simulation. The BioModels usually contain deterministic rate functions instead of stochastic rate constants, and thus, a decision was made to set the stochastic rate constants of all reactions to 1.0. This is technically justified as

it turns the deterministic models into stochastic ones, and because the property analysis is performed upon the *unweighted* dependency graphs derived from these models (i.e. dependency graphs independent of reaction rates).

As the BioModels are limited in size, we were interested to test the accuracy of the predictors algorithmic performance when presented with a much larger model. We used the Escherichia coli AI-2 quorum signal circuit from Li et al. (*25*). Whilst this model only contains 25 reactions and 21 species, we scaled it up by instantiating it on each point of a hypothetical two-dimensional lattice. We generated 10×10 and 100×100 models, adding transport reactions between adjacent lattice points to incorporate quorum signal molecule transfer (see Table 4).

Table 4: Reaction and species graph sizes for different versions of the stochastic Escherichia coli AI-2 quorum signal circuit model from Li et al. The model size is the number of points on a 2D lattice the model was instantiated on.

model size	**reactions**	**species**
1×1	25	21
10×10	2860	2100
100×100	289600	210000

3.3 Topological Analysis

Ssapredict performs a model property analysis and uses the results to predict the fastest SSA for that model. After parsing the model, a reaction dependency graph and species dependency graph are generated. In a reaction dependency graph, each vertex corresponds to a unique reaction, hence the number of vertices in a reaction dependency graph is equal to the number of reactions in the model. A directed edge is placed from vertex V_i to vertex V_j if the firing of reaction R_i changes the propensity of reaction R_j. In a species dependency graph, each vertex corresponds to a unique species, and so the number of vertices in a species dependency graph is equal to the number of species in the model. A directed edge is drawn from vertex V_i to vertex V_j if for any reaction species S_i is a reactant and species S_j is a product. Any duplicate edges are removed from each graph.

54 properties were generated for each of the reaction and species dependency graphs of a model. An additional property, reaction stiffness ratio, was also calculated bringing the total number of properties to 109. For some models certain properties were not possible to compute, for example, when a division by zero occurred. We replaced all missing values with zeros. Nine model properties were found to be constant for all models and therefore would be of no use as performance indicators and thus were removed from the data set, resulting in 100 model properties available for analysis. *Ssapredict* analyses a restricted set of 32 *fast* (computational complexity $\leq O(V + E)$) properties ensuring that the graph analysis completes in a timely manner (see Table 5).

3.4 Algorithm Performance

The performance metric used to measure algorithmic computational speed was reactions per second (*rps*) of CPU time. Rps allows one to compare algorithm performance in a manner that ignores simulation run time. This means that algorithm performance can be compared between two models that take vastly differing amounts of time to execute. Using rps also improves comparative accuracy, if we wish to run an algorithm for x seconds, and measure how many reactions are executed, the amount of time elapsed would almost certainly not be *exactly* x seconds, but a number extremely close to x seconds. If this was compared to another run of x seconds, neither run would be exactly the same amount of time, and thus a comparison in this manner would lose accuracy. Dividing the number of reactions executed by the exact simulation time to get a result in rps leaves us with a value that is appropriate for comparison. All runs were executed on a single core of an otherwise idle benchmarking machine that possessed an Intel i7 2600K CPU with 16GB RAM and was running Ubuntu 11.04. The large amount of RAM available and size of models involved meant that all simulations could be run in memory and thus avoid performance deteri-

Table 5: Summary of model topological properties analysed. Complexity relates to *worst case time complexity* for the computation of the propensity, where V is vertices, E is edges, and d is the average node degree. Properties marked with † have constant or linear scaling and are used for the restricted set of *fast* properties.

computational complexity	**graph property**
$O(1)^\dagger$	number of edges, number of vertices, density of graph
$O(V)^\dagger$	min/mean/max outgoing edges, min/mean/max incoming edges, min/mean/max all edges
$O(V+E)^\dagger$	weakly connected components, articulation points, bi-connected components, reciprocity of directed graph
$O(VE)$	average geodesic length, longest geodesic length, min/mean/max outgoing closeness, min/mean/max incoming closeness, min/mean/max closeness in undirected graph, min/mean/max betweenness, min/mean/max betweenness in undirected graph, min/mean/max edge betweenness, min/mean/max edge betweenness undirected graph
$O(V(V+E))$	min/mean/max shortest path in undirected graph, min/mean/max shortest incoming path, min/mean/max shortest outgoing path
$O((V+E)^2)$	girth of undirected graph
$O(Vd^2)$	transitivity of graph vertices, average local transitivity
$O(V^4)$	min edge connectivity
$O(V^5)$	min vertex connectivity

orations caused by memory paging.

Each of the BioModels were executed for 10 seconds of CPU time for all 9 algorithms. Each algorithm was run 10 times on each model, hence a total of 90 rps values for each model. Because the BioModels were void of parameters such as simulation time and species amounts, it was decided that 10 seconds of CPU time for each model/algorithm combination would be enough to determine an accurate result for algorithm performance.

3.5 Prediction Methods

We used two variants of a random predictor, a classifier based on a set of linear regression estimators trained separately for each algorithm, logistic regression (*26*), support vector classifier with linear kernel (*27*) and a nearest neighbour classifier (*28*) using a vote of 5 nearest models. For each predictor we performed a 10-fold cross-validation experiment and measured the mean accuracy and standard deviation.

The two random predictors used different amounts of information. First was a blind random predictor which assumed that each algorithm is equally probable to perform best. The probability of such a blind guess to be correct is equal to $\frac{1}{9}$ (see Equation 1).

$$p = \sum_i w_i p_i = \frac{1}{9} \sum_i p_i = \frac{1}{9} \qquad (1)$$

The second random predictor assumed that each algorithm is as probable as it was observed in the training set. Then, roulette wheel selection was used to make a prediction. In the ideal case, the informed random guess would assign a weight equal to the true probability of winning for each algorithm (see Equation 2). Given

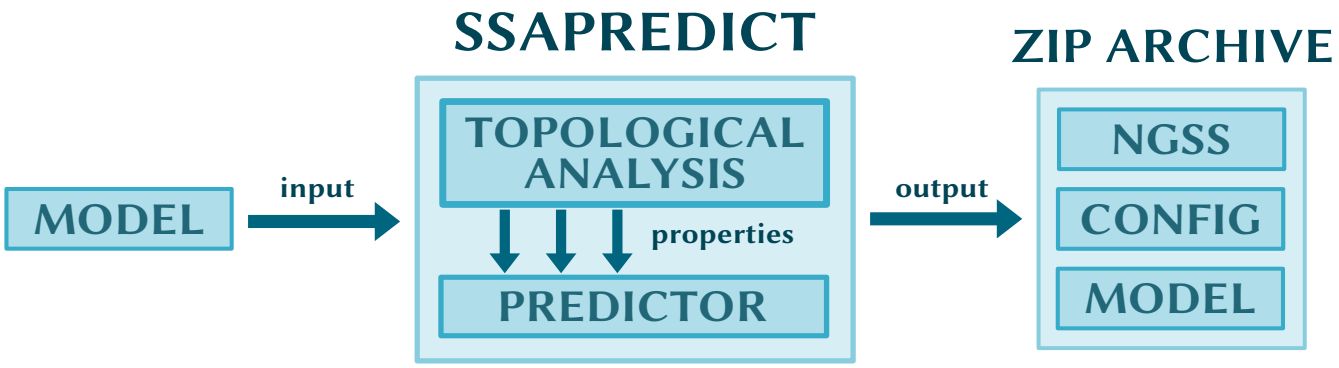

Figure 9: Structural diagram of *ssapredict* architecture and work-flow.

the distribution of winners in our benchmark we expect the probability of a correct guess to be three times greater than in the case of the blind guess.

$$p = \sum_i w_i p_i = \sum_i p_i^2 \simeq 0.27 \qquad (2)$$

3.6 Application Architecture

The *ssapredict* web application is built with python using the web2py framework (*29*). Figure 9 visualises the structure and work-flow of *ssapredict*. Models are uploaded to *ssapredict* in SBML (*20*) format and parsed using libSBML (*30*). After producing dependency graphs from the model data, a model topological analysis is performed. The model property analysis software is written in C++ with performance as a design priority and calls functions from the igraph library (*31*) to compute the graph properties.

The application then performs fast property analysis of the supplied model using the set of 32 computationally inexpensive properties. The result of this model analysis is delivered to a linear SVC classifier which makes a prediction of the fastest algorithm to simulate the model. Linear SVC was the best performing classification method identified by the cross-validation experiments and is trained with the algorithmic performance data and fast set of properties for all models. The linear SVC classifier is written in Python using scikit-learn (*32*).

After a prediction has been made, *ssapredict* allows users to simulate their models with the predicted fastest algorithm. The simulator *ngss* is written in C++ with an emphasis on performance and released under the terms of the GNU General Public Licence. *Ngss* is portable and will operate on GNU/Linux, Mac OS X and Windows platforms. *Ngss* is distributed such that no extra dependencies need to be installed by the user. *Ngss* supports OpenMP, allowing parallel runs (stochastic trajectories) on multi-core computers. *Ssapredict* simulation functionality is provided to the user through a zip archive that contains the uploaded model and simulator executable for their platform. An auto-generated simulation parameter file that configures *ngss* is also contained in the archive. By simply extracting the files and entering a one line command (instructions provided in archive), the simulator will generate stochastic trajectories as time-series in CSV format.

Acknowledgement The authors thank Jamie Twycross and Jonathan Blakes for their collaboration on models processing. This work was supported by the Engineering and Physical Sciences Research Council [EP/I031642/1, EP/J004111/1, EP/L001489/1].

4 Author Contributions

D. Sanassy developed the *ngss* stochastic simulator. D. Sanassy and P. Widera developed the *ssapredict* web application. P. Widera, D. Sanassy and N. Krasnogor analysed the data. N. Krasnogor conceived the meta-stochastic simulator concept. D. Sanassy, P. Widera and N. Krasnogor wrote the manuscript.

References

1. Blakes, J., Twycross, J., Romero-Campero, F. J., and Krasnogor, N. (2011) The Infobiotics Workbench: an integrated in silico modelling platform for Systems and Synthetic Biology. *Bioinformatics 27*, 3323–3324.

2. Twycross, J., Band, L., Bennett, M., King, J., and Krasnogor, N. (2010) Stochastic and deterministic multiscale models for systems biology: an auxin-transport case study. *BMC Systems Biology 4*, 34.

3. Cao, H., Romero-Campero, F., Heeb, S., Cámara, M., and Krasnogor, N. (2010) Evolving cell models for systems and synthetic biology. *Systems and Synthetic Biology 4*, 55–84.

4. Romero-Campero, F. J., Twycross, J., Cámara, M., Bennett, M., Gheorghe, M., and Krasnogor, N. (2009) Modular Assembly of Cell Systems Biology Models Using P Systems. *International Journal of Foundations of Computer Science 20*, 427–442.

5. Gillespie, D. T. (1976) A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics 22*, 403–434.

6. Gillespie, D. T. (2001) Approximate accelerated stochastic simulation of chemically reacting systems. *The Journal of Chemical Physics 115*, 1716–1733.

7. Gibson, M. A., and Bruck, J. (2000) Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels. *The Journal of Physical Chemistry A 104*, 1876–1889.

8. McCollum, J. M., Peterson, G. D., Cox, C. D., Simpson, M. L., and Samatova, N. F. (2006) The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior. *Computational Biology and Chemistry 30*, 39–49.

9. Li, H., and Petzold, L. *Logarithmic direct method for discrete stochastic simulation of chemically reacting systems.*; 2006.

10. Cao, Y., Li, H., and Petzold, L. (2004) Efficient formulation of the stochastic simulation algorithm for chemically reacting systems. *The Journal of Chemical Physics 121*, 4059–4067.

11. Slepoy, A., Thompson, A. P., and Plimpton, S. J. (2008) A constant-time kinetic Monte Carlo algorithm for simulation of large biochemical reaction networks. *The Journal of Chemical Physics 128*, 205101.

12. Ramaswamy, R., Gonzalez-Segredo, N., and Sbalzarini, I. F. (2009) A new class of highly efficient exact stochastic simulation algorithms for chemical reaction networks. *The Journal of Chemical Physics 130*, 244104.

13. Sanft, K. R., Wu, S., Roh, M., Fu, J., Lim, R. K., and Petzold, L. R. (2011) StochKit2: software for discrete stochastic simulation of biochemical systems with events. *Bioinformatics 27*, 2457–2458.

14. Hoops, S., Sahle, S., Gauges, R., Lee, C., Pahle, J., Simus, N., Singhal, M., Xu, L., Mendes, P., and Kummer, U. (2006) COPASI — a COmplex PAthway SImulator. *Bioinformatics 22*, 3067–3074.

15. Mauch, S., and Stalzer, M. (2011) Efficient formulations for exact stochastic simulation of chemical systems. *IEEE/ACM Transactions on Computational Biology and Bioinformatics 8*, 27–35.

16. Chandran, D., Bergmann, F., and Sauro, H. (2009) TinkerCell: modular CAD tool for synthetic biology. *Journal of Biological Engineering 3*, 19.

17. Roy, S. (2012) Systems biology beyond degree, hubs and scale-free networks: the case for multiple metrics in complex networks. *Systems and Synthetic Biology 6*, 31–34.

18. Gillespie, D. T. (1977) Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry 81*, 2340–2361.

19. Li, C., Donizelli, M., Rodriguez, N., Dharuri, H., Endler, L., Chelliah, V., Li, L., He, E., Henry, A., Stefan, M. I., Snoep, J. L., Hucka, M., Le Novère, N., and Laibe, C. (2010) BioModels Database: An enhanced, curated and annotated resource for published quantitative kinetic models. *BMC Systems Biology 4*, 92.

20. Hucka, M. et al. (2003) The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics 19*, 524–531.

21. Cao, Y., Gillespie, D. T., and Petzold, L. R. (2005) The slow-scale stochastic simulation algorithm. *The Journal of Chemical Physics 122*, 014116.

22. Rathinam, M., Petzold, L. R., Cao, Y., and Gillespie, D. T. (2003) Stiffness in stochastic chemically reacting systems: The implicit tau-leaping method. *The Journal of Chemical Physics 119*, 12784–12794.

23. Gillespie, D. T. (2007) Stochastic simulation of chemical kinetics. *Annual Review of Physical Chemistry 58*, 35–55.

24. Cao, Y., Gillespie, D. T., and Petzold, L. R. (2006) Efficient step size selection for the tau-leaping simulation method. *The Journal of Chemical Physics 124*, 044109.

25. Li, J., Wang, L., Hashimoto, Y., Tsao, C.-Y., Wood, T. K., Valdes, J. J., Zafiriou, E., and Bentley, W. E. (2006) A stochastic model of Escherichia coli AI-2 quorum signal circuit reveals alternative synthesis pathways. *Molecular systems biology 2*, 67.

26. Yu, H.-F., Huang, F.-L., and Lin, C.-J. (2011) Dual coordinate descent methods for logistic regression and maximum entropy models. *Machine Learning 85*, 41–75.

27. Chang, C.-C., and Lin, C.-J. (2011) LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology 2*, 27:1–27:27.

28. Wu, X., Kumar, V., Ross Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G., Ng, A., Liu, B., Yu, P., Zhou, Z.-H., Steinbach, M., Hand, D., and Steinberg, D. (2008) Top 10 algorithms in data mining. *Knowledge and Information Systems 14*, 1–37.

29. Di Pierro, M. (2011) Web2Py for scientific applications. *Computing in Science and Engineering 13*, 64–69.

30. Bornstein, B. J., Keating, S. M., Jouraku, A., and Hucka, M. (2008) LibSBML: an API Library for SBML. *Bioinformatics 24*, 880–881.

31. Csardi, G., and Nepusz, T. (2006) The igraph software package for complex network research. *InterJournal Complex Systems*, 1695.

32. Pedregosa, F. et al. (2011) Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research 12*, 2825–2830.