Approximate probabilistic analysis of biopathway dynamics

Bing Liu, Andrei Hagiescu, Sucheendra K. Palaniappan, Bipasa Chattopadhyay, Zheng Cui, Weng-Fai Wong and P.S. Thiagarajan∗
Department of Computer Science, National University of Singapore, 117417 Singapore

ABSTRACT
Motivation: Biopathways are often modeled as systems of ordinary differential equations (ODEs). Such systems will usually have many unknown parameters and hence will be difficult to calibrate. Since the data available for calibration will have limited precision, an approximate representation of the ODEs dynamics should suffice. One must however be able to efficiently construct such approximations for large models and perform model calibration and subsequent analysis.

Results: We present a GPU-based scheme by which a system of ODEs is approximated as a dynamic Bayesian network (DBN). We then construct a model checking procedure for DBNs based on a simple probabilistic linear time temporal logic. The GPU implementation considerably extends the reach of our previous PC-cluster based implementation (Liu et al., 2011b). Further, the key components of our algorithm can serve as the GPU kernel for other Monte Carlo simulations based analysis of biopathway dynamics. Similarly, our model checking framework is a generic one and can be applied in other systems biology settings.

We have tested our methods on three ODE models of biopathways: the EGF-NGF pathway, the segmentation clock network and the MLC-phosphorylation pathway models. The GPU implementation shows significant gains in performance and scalability while the model checking framework turns out to be convenient and efficient for specifying and verifying interesting pathways properties.

Availability: The source code is freely available at http://www.comp.nus.edu.sg/~rpsysbio/pada-gpu/

Supplementary information: Supplementary data are available at Bioinformatics online.

1 INTRODUCTION
The modeling and analysis of biopathways dynamics is a core activity in systems biology. A standard approach is to view a biopathway as a network of biochemical reactions and to model the network as a system of ordinary differential equations (ODEs) (Aldridge et al., 2006). Since biopathways often involve a large number of reactions, the corresponding ODE systems will not admit closed form solutions and one will have to resort to numerical simulations. However the ODE systems will often contain many unknown parameters (rate constants and initial concentration levels) which will first have to be estimated using meager data of limited precision. Consequently, for large pathways model construction and analysis are difficult problems. With this as motivation, a probabilistic approximation method was developed in Liu et al. (2009) by which an ODE system is reduced to a dynamic Bayesian network (DBN). Parameter estimation followed by sensitivity analysis is then carried out on this simpler model using standard Bayesian inference techniques. This method is promising in terms of efficiency, accuracy and applicability (Liu et al., 2011b,a). Our goal here is to extend this scheme in two significant ways. The core features of these two extensions are of independent interest and can be deployed in other settings involving biopathways models.

The first extension is a parallel implementation of the DBN approximation scheme using Graphical Processing Units (GPUs). It is computationally intensive to construct the DBN from a system of ODEs. In our experience, a single PC is hopelessly inadequate while even a PC-cluster quickly runs into scalability issues. On the other hand, a supercomputing facility may not be available or affordable. In comparison, GPUs provide an excellent combination of cost and performance. However, not all algorithms map well onto a GPU platform due to its memory hierarchy. Specifically, one must carefully balance parallelism with memory accesses to obtain good performance.

In our DBN construction -explained in more detail in the next section- a computationally intensive phase is the generation of a large number of trajectories using numerical simulations. This can be done in parallel and hence the GPU platform is a natural choice. However, each variable can appear in multiple equations. Hence to generate a trajectory one must, in principle, access all the equations in each integration step. Further, the threads generating the trajectories will have to record a good deal of intermediate information to construct the conditional probability tables (CPTs) of the DBN. For large ODE systems the intermediate data generated will be too large to be stored in the fast local memory. One will instead have to use the slow global memory for this purpose. This can lead to a severe degradation in performance. To get around this we create a heterogeneous pool of threads in which each trajectory is computed in a distributed fashion while a second group of threads manage the data movement. There are other settings in which a set of trajectories (Lüdtke et al., 2008; Li et al., 2010) is generated which is then subjected to statistical analysis to derive system properties. In such applications too, our compilation strategy will likely lead to high performance GPU implementations.

∗ to whom correspondence should be addressed
The second major contribution of the paper is a probabilistic model checking procedure for DBNs. We use a simple probabilistic variant of linear time temporal logic (LTL) (Pnueli, 1977) in which the atomic propositions are of the form \((X, v) \leq c\) or \((X, v) \geq c\) where \(X\) is a finite valued random variable corresponding to a node in the DBN and \(c\) is rational number in \([0, 1]\). The assertion \((X, v) \leq c\) says that the probability of the random variable \(X\) currently having the value \(v\) is less than or equal to \(c\); similarly for the assertion \((X, v) \geq c\). The rest of the syntax is standard. Though probability enters the logic only via atomic propositions it turns out that one can still express many interesting dynamical properties. A key component of our model checking procedure is a Bayesian inferences algorithm called the factored frontier algorithm (Murphy and Weiss, 2001) that is used to approximately determine the truthhood of the atomic propositions. We do this approximately since exact inference is computationally infeasible for large DBNs. The accuracy of this inferring procedure can be improved -at additional computational cost- by using a parametrized variant of the factored frontier algorithm that we have recently developed (Palaniappan et al., 2011) (More details can be found in the Supplementary Information.) Again, this simple and novel version of probabilistic verification can be applied to other related settings (Langmead et al., 2006).

Figure 1 shows the overall framework in which we have carried our work. As indicated, our approximation procedure -implemented on a GPU platform- will often yield an uncalibrated DBN due to the presence of unknown parameters. We then perform parameter estimation using Bayesian inferencing to obtain a calibrated model. One can then carry out -on a PC platform- tasks such as sensitivity analysis and probabilistic verification. At present these tasks are performed on a PC platform. In Liu et al. (2011b) we show how parameter estimation followed by sensitivity analysis is performed using the Bayesian inferencing. Hence we do not deal with this here and instead focus on the new analysis method, namely, probabilistic verification.

We have tested the GPU implementation on ODE models of the EGF-NGF pathway (Brown et al., 2004), the segmentation clock network (Goldbeter and Pourquie, 2008) and the thrombin-dependent MLC phosphorylation pathway (Maeda et al., 2006). We obtained these models from the BioModels database (Le Novere et al., 2006). Although the parameter values for these models are known, in each case we set a subset of the parameters as unknown to mimic realistic biopathways models. This considerably increases the computational demands placed on the DBN construction. For each model, we constructed a DBN by generating 3 million trajectories. We then compared the performance of the GPU implementation with that of a 10-CPU cluster. The EGF-NGF model consists of 32 differential equations and 48 kinetic parameters from which 20 were set to be unknown. The GPU implementation ran 26 times faster. The segmentation clock network model consists of 22 differential equations and 75 kinetic parameters out of which 40 were set to be unknown. In this case the GPU implementation ran 32 times faster. The third model was that of the thrombin-dependent MLC phosphorylation pathway consisting of 105 differential equations and 197 kinetic parameters out of which we set 164 to be unknown. In this case, the GPU implementation took 38 hours while the cluster implementation turned out to be infeasible. Even for 30000 trajectories it took 37 hours and hence for a 3 million trajectories, it would have taken around 5 months. Further, the 30000-trajectories based DBN that the PC-cluster produced was of poor quality.

We also tested our model checking procedure on the DBN approximations of these three pathway models. For each model we formulated a number of dynamical properties and verified them to be true or false. It only took less than 1 second on a PC to verify each property.

Turning to related work, an excellent overview of biopathways dynamics and formalisms for studying their behaviors can be found in Klipp et al. (2005). Modeling and analysis of biopathways using ODEs is well-established domain (Aldridge et al., 2006; Sreenath et al., 2008). As mentioned earlier, our approximation was presented in Liu et al. (2011b) and has been applied in a concrete biological setting (Liu et al., 2011a). As for related GPU applications, a survey of hardware accelerators for biocomputing including GPUs is presented in Dematte and Prandi (2010). Of particular relevance is the Python language based package called cuda-sim reported in Zhou et al. (2011). This package enables accelerated simulations of biochemical network models on GPUs. Apart from ODEs based models, the cuda-sim package also supports models based on stochastic differential equations as well as Markov Jump processes. We consider this work to be orthogonal to ours in the sense its main focus is on the simulations of ODEs. In our setting, generating a large number of trajectories through numerical integration is just one component of the larger task of constructing the DBN approximation. At present we do not have a GPU based implementation of the parameter estimation, sensitivity analysis and probabilistic verification procedures. In this connection, the work on the GPU implementation of the sum-of-product algorithm (Silberstein et al., 2008) for DBNs is promises to be very relevant.

As for the second component of our work, model checking based on temporal logics is a well-established domain. It was founded in (Pnueli, 1977) through the seminal work of Amir Pnueli where he proposed Linear Time Temporal Logic (LTL) as the basis for reasoning about the behavior of programs. An ideal starting point for delving into this field is Clarke et al. (1999) which presents the basic family of temporal logics as well the automated verification procedure called model checking. This framework was extended to probabilistic systems -based on discrete time Markov chains- using the probabilistic temporal logic called PCTL in Hansson and Jonsson (1994). It is worth noting that our specification logic PBL is simple fragment of PCTL. Overviews of this active area of research can be found for instance in Kwiatkowska et al. (2010); Legay et al. (2010).
A number of modeling formalisms (Danos et al., 2007; Kwiatkowska et al., 2008; Ciochetta and Hillston, 2009; Henzinger et al., 2010) for biochemical networks are based on Continuous Time Markov Chains (CTMCs). Model checking the properties of these stochastic models using probabilistic temporal logics such as PCTL and its variants is being actively pursued by a number of researchers (Clarke et al., 2008; Kwiatkowska et al., 2008; Gong et al., 2010; Li et al., 2010). Of particular interest in our context is Donaldson and Gilbert (2008) which defines the probabilistic LTL with numerical constraints (PLTLc) and verifies properties of ODE models through Monte Carlo integration. Here instead we carry out Monte Carlo simulations once to construct a DBN approximation. One can then repeatedly perform probabilistic verification of properties expressed in PBL using Bayesian inferencing. In Clarke et al. (2008), a technique called statistical model checking is used to verify properties of a CTMC model expressed in a logic called bounded LTL (BTL) which is basically LTL interpreted over finite sequences. They stochastically simulate a CTMC model for a finite number of runs and use BTL to check each simulation run. They then perform statistical tests to verify that this property meets a required probability bound. Our logic PBL is also interpreted over finite sequences. However it is a probabilistic logic in that the atomic propositions make probabilistic assertions. Further, the logic is interpreted over the single sequence of marginal distributions generated by the DBN (as explained in section 4). Finally, the work reported in Langmead et al. (2006) also carries out DBN based model checking. However, the DBN is first converted to a data structure called a Multi-Terminal Binary Decision Diagram (MTBDD) and then existing probabilistic model checking algorithms based on PCTL are applied. Additional background information on these topics can be found in the Supplementary Information.

In the next section, we explain how a DBN is constructed from a system of ODEs. In section 3, we present the main features of our GPU implementation. In the subsequent section we introduce our probabilistic variant of LTL and the associated model checking procedure. In section 5 we present our experimental results. The final section summarizes and discusses possible extensions of the work presented here.

2 THE DBN APPROXIMATION

The dynamics of a biopathway is often modeled as a system of ODEs with one equation of the form \( \frac{dx}{dt} = f(x, p) \) for each molecular species \( x \) in the pathway. Here \( f \) describes the kinetics of the reactions that produce and consume \( x \) and \( p \) are the molecular species taking part in these reactions while \( p \) are the rate constants governing these reactions. For large pathways, this ODE system which will typically have many unknown parameters will be difficult to calibrate and analyze. To get around this an approximation scheme was developed in Liu et al. (2009) through which a system of ODEs can be reduced to a dynamic Bayesian network. Our goal here is to briefly explain this method. The technical details can be found in Liu et al. (2011b).

First we assume the states of the system are observed only at a finite number of time points, \( \{0, 1, \ldots, T\} \). Next, the range of each variable \( x_i \) (rate constant \( r_j \)) is partitioned into a set of intervals \( I_i \). (1) Both these discretizations are motivated by the fact that experimental data will be available only for a finite set of time points and this data will be of limited precision.

\[
S + E \xrightarrow{k_{-3}} ES \xrightarrow{k_3} E + P
\]

(a) Reaction network

Next, the initial values of the variables as well as the rate constants are assumed to be distributions (usually uniform) over certain of these intervals. We then sample the initial states of the system according to this distribution sufficiently many times, and generate a trajectory by numerical integration for each sampled initial state. The resulting set of trajectories is then treated as an approximation of the dynamics of ODE system.

To handle unknown rate constants we assume that the minimum and maximum values of these constants are known. We then partition these ranges of values also into a finite numbers of intervals, and fix a uniform distribution over all the intervals. After building the DBN, we use a Bayesian inference based technique to perform parameter estimation to complete the construction of the model. However, unlike the variables, once the initial value of an unknown rate constant has been sampled, this value will not change during the generation of a trajectory. Naturally different trajectories can have different initial values for an unknown rate constant.

A key idea is to compactly store the generated set of sequences as a dynamic Bayesian network. This is achieved by means of a simple counting procedure that exploits the network structure. In order to keep the focus on the approximation procedure we give only an informal description of DBNs here and defer a precise presentation to section 4.

A DBN consists of a directed acyclic graph where the nodes are grouped into layers with each layer representing a time point (Murphy, 2002). The nodes in layer \( t - 1 \) will be connected to the nodes in the layer \( t \) in the same way as \( t \) ranges from 1 to \( T \). Each node will have a random variable associated with it. In our setting, there will be one random variable \( x_i^t \) corresponding to each variable \( x_i \) (unknown rate constant \( r_j \)) in the interval in which value of \( x_i \) falls at time \( t \). Further, for each unknown rate constant \( k \), we add the equation \( \frac{dk}{dt} = 0 \) to capture the fact that once the value of \( k \) has been sampled, this value will not change during the numerical integration of a trajectory.

\[ Pr(x_i^t) \], the set of parent nodes of \( x_i^t \) is determined as follows. The node \( x_i^{t-1} \) will be in \( Pr(x_i^t) \) if \( x_k \) appears in the equation for \( x_i \) or \( x_k = x_i \). On the other hand, \( r_j^{t-1} \) will be the only parent of the node \( r_j^t \) in case \( r_j \) is an unknown rate constant.

In Figure 2, we show a simple enzymatic reaction network, its ODE

(b) ODE model

(c) DBN model

Fig. 2. (a) The enzyme catalytic reaction network. (b) The ODE model. (c) The DBN approximation.
model and the structure of its DBN approximation. In this example, we have assumed that \( k_3 \) is the only unknown parameter.

As indicated in Figure 2(c), each node will also have a conditional probability table (CPT) associated with it to specify the local probabilistic dynamics. A typical entry in the CPT of \( x'_2 \) will be of the form \( \text{Pr}(x'_2|I, z_1^{-1}, z_2^{-1}) = I_1, z_1^{-1} = 1, z_2^{-1} = 2, \ldots, z_l^{-1} = l) = p \) with \( Pa(x'_2) = \{z_1^{-1}, z_2^{-1}, \ldots, z_l^{-1}\} \). Such an entry means that \( p \) is the probability that the value of \( x'_2 \) falls in the interval \( I \) at time \( t \), given that the value of \( z_l \) was in \( I_l \) at time \( t - 1 \) for each \( z_l^{-1} \) in \( Pa(x'_2) \). The probability \( p \) is calculated through simple counting. Suppose \( N \) is the number of generated trajectories. We record the number of the trajectories from this collection for which their value of \( z_l \) fell in the interval \( I_l \) for each \( z_l \) in \( \{z_1, z_2, \ldots, z_l\} \) at time \( t \). Suppose this number is \( J \). We then determine for how many of these \( J \) trajectories, the value of \( x'_2 \) fell in the interval \( I \) at time \( t \). If this number is \( J' \), then \( p \) is set to be \( \frac{J'}{J} \).

If \( k \) is unknown, in the CPT of \( k' \) we will have \( \text{Pr}(k'|I = I') = 1 \) if \( I = I' \) and \( \text{Pr}(k'|I = I' | k_l^{-1} = I_l' = 0) = 0 \) otherwise. This is because the sampled initial value of \( k \) does not change during numerical integration. Suppose \( k \) appears on the right hand side of the equation for \( x \) and \( Pa(x) = \{z_1^{-1}, z_2^{-1}, \ldots, z_l^{-1}\} \) with \( z_l^{-1} = k^{-1} \). Then for each choice of interval values for nodes other than \( k \) in \( Pa(x) \) and for each choice of interval value \( I \) for \( k \) there will be an entry in the CPT of \( x' \) of the form \( \text{Pr}(x'_l = I | z_l^{-1} = I_1, z_2^{-1} = I_2, \ldots, k = I) = p \). This is so since we will sample for all possible initial interval values for \( k \) and \( k_l^{t-1} = k^{-1} \).

In this sense the CPTs record the approximated dynamics for all possible combinations of interval values for the unknown rate constants. These features are illustrated in Figure 2(c) for the unknown rate constant \( k_3 \).

By performing parameter estimation on the resulting uncalibrated DBN one can obtain a calibrated DBN in which each parameter will have a specific interval value assigned to it.

### 3 MAPPING TO A GPU ARCHITECTURE

We now describe how our approximation algorithm is mapped onto the NVIDIA’s Fermi platform. In this platform each GPU unit consists of 16 streaming multiprocessors (SMs for short). Each SM has 32 arithmetic cores divided into two groups of 16 each for scheduling purposes (Glaskowsky, 2009). Threads are grouped into warps (Single Instruction Multiple Data) manner. Thus in any cycle -subject to some mild constraints- two half-warps can be executing in a SM. However, the threads belonging to a half warp must execute the same instruction. If not, they must be serialized. Pipelining offers an additional dimension of parallelism. Loosely speaking, it takes about 22 cycles to execute each instruction. Through pipelining, when the \( i + \frac{1}{2}\)th processing cycle of the instructions belonging to a warp starts executing, the system can schedule the execution of the \( i\)th processing cycle of the instructions belonging to a different wrap. As for the memory hierarchy, each SM has 32K registers of 32 bits length and each thread in an SM can be allocated up to 64 of these registers. Each SM has 48KB of shared fast memory which the threads belonging to the SM can use to store data and synchronize with each other. Additional data must be stored in the slow-off-chip global memory of size 2GB. This is also the only medium through which threads belonging to different SMs can synchronize. In addition there is a small \( L1 \) cache of size 16KB belonging to each SM and an \( L2 \) cache of size 768KB that is shared by all the SMs. Each SM computes a set of trajectories and records the number of times these trajectories hit the intervals of values of the variables at different time points. This binned information is stored in a specific area of the global memory which will be summed up to produce the CPTs of the DBN. We now describe how the computation within a single SM is orchestrated according to the scheme shown in Figure 3.

Starting from an initial state at \( t = 0 \), for each time interval \( \Delta t \), the new value of a variable \( x \) is determined by applying numerical integration using the current values of the variables and the values of the rate constants appearing in the ODE for \( x \) as well as the current value of \( x \). Due to the coupling between the variables, the entire front of the new values of all the variables must be computed at each time step per trajectory. If we naively allocate as many threads as possible to each SM with each thread computing a trajectory then their memory requirements will exceed the size of the (fast) local memory of the SM.

To get around this we partition the set of equations into blocks and allocate each block to a thread. Thus a single trajectory will be computed by a set of threads \( C \). Each member of \( C \) will handle a different block of equations and compute the new values of the variables appearing on the left hand sides of these equations. Each thread will read from the local memory of the SM the current values of the variables \( v \) and the rate constants \( p \) appearing in the equations allotted to it. The \( \Delta t \) changes during a time step are computed in parallel and are stored back to the local memory. The vector of variables \( v \) is then updated. This process is applied iteratively for each time step.

We must also perform the binning steps to record the number of times the threads hit the various intervals of values of the variables (and unknown rate constants). This is required for constructing the CPTs of the DBN. Accordingly, the vector \( v \) is replicated as \( v \). The binning process executes in parallel, during the subsequent \( \Delta t \) iteration, using the memory access threads \( (M) \), which will store the results in a large table located in global memory. This will ensure that the numerical integration continues during the binning process.

Many copies of the \( C \) and \( M \) groups of threads will be assigned to an SM. How they are scheduled is guided by the hardware organization of the SM explained above. In particular, we allot the \( C \) threads belonging to each trajectory to the warps so that threads executed together in the same warp process the same block of equations corresponding to different trajectories. This implies that threads

![Fig. 3. Concurrent execution of trajectories inside an SM.](http://bioinformatics.oxfordjournals.org/Downloaded from National University of Singapore on April 12, 2012)
computing different blocks corresponding to the same trajectory are assigned to different warps.

We next turn to the issue of partitioning the set of equations into blocks with each block handled by a thread in $C$. The compiler converts each equation into a set of memory loads and stores and a set of arithmetic operations. To achieve good balance, we have created a simple timing model that assigns a weight to each operation that appears in each equation. We then distribute the equations so that the corresponding weight of the operators appearing among the $C$ threads is balanced.

Our compilation strategy leads to a better utilization of the fast local memory in comparison to the naive implementation in which each thread computes a trajectory and the CUDA compiler decides how many parallel threads should be launched. Our experiments using an initial implementation showed that for one of the case studies reported in section 5 (the EGF-NGF model) it achieves approximately a 40% improvement over the naive solution. We have not however carried out a systematic and detailed comparison so far.

It is also worth pointing out that there will be ODE systems on which our implementation will not do much better than the naive implementation. This is so since structure of the DBN is determined by the couplings between the variables in the ODE system. Hence our solution will not do much better for networks in which the graph denoting the couplings between the variables is not sparse but instead has many nodes that have a large number of parent nodes. This situation will arise when the biochemical network under study instead has many nodes that have a large number of parent nodes. To generate the initial states, we used a Mersenne twister algorithm based on the MT19937 random number generator (Matsumoto and Nishimura, 1998), running in each of the $C$ threads. We used a fourth order Runge-Kutta algorithm for the numerical integration process (Hindmarsh, 1983). We have extended the implementation described above to multiple GPUs running in parallel. Each GPU computes independently its portion of the trajectories and records the results in a table in its own global memory. Once the computation has finished, these tables are transferred over the network to the master host that combines them to form the CPTs. More details regarding our GPU implementation can be found in the Supplementary Information.

4 PROBABILISTIC MODEL CHECKING

Once the DBN approximation has been constructed many analysis tasks can be carried out efficiently, such as parameter estimation and sensitivity analysis via Bayesian inferencing (Liu et al., 2011b). Here we formulate a new analysis method based on probabilistic model checking (FF for short) will play a key role in our model checking procedure. Hence we start with DBNs and then describe FF.

4.1 The Factored Frontier algorithm

We fix an ordered set of $n$ random variables $\{X_1, \ldots, X_n\}$ and let $i$, $j$ range over $\{1, 2, \ldots, n\}$. We denote by $X$ the tuple $(X_1, \ldots, X_n)$. The random variables are assumed to take values from the finite set $V$ of cardinality $K$. We let $x_i, v_i, v_i$ to denote a value taken by $X_i$. In our application the random variables will correspond to the variables (and unknown rate constants) appearing in the ODE model. Our dynamic Bayesian networks will be time-variant but with a regular structure. They will be unrolled over a finite number of time points. Further, there will be no hidden variables (Murphy and Weiss, 2001).

Formally, a DBN is a structure $\mathcal{D} = \{X, T, Pa, \{C_i\}\}$ where,

- $T$ is a positive integer with $t$ ranging over the set of time points $\{0, 1, \ldots, T\}$ is the set of random variables. As usual, these variables will be identified with the nodes of the DBN. $X_1$ is the instance of $X_t$ at time slice $t$.
- $Pa$ assigns a set of parents to each node and satisfies: (i) $Pa(X_1) = \emptyset$ if $X_1$ is a root in $\mathcal{D}$ (ii) $Pa(X_t') \subseteq Pa(X_t')$ for some $t$ then $X_t' = X_t$. If $X_t' \in Pa(X_t')$ for every $t' \in \{1, 2, \ldots, T\}$, $\mathcal{D}$ is a first-order DBN. Thus the way nodes at the $t$th time slice are connected to nodes at the $t$th time slice remains invariant as $t$ ranges over $\{1, 2, \ldots, T\}$.
- $C_i$ is the Conditional Probability Table (CPT) associated with node $X_i$ specifying the probabilities $Pr(X_1' \mid Pa(X_1'))$. Suppose $Pa(X_1') = \{X_1', X_2', \ldots, X_m'\}$ and $(x_1, x_2, \ldots, x_m) \in V^m$. Then a CPT entry of the form $C_i(x_1, x_1', X_2, X_2', \ldots, X_m)$ is $1$. As usual we require, $\sum_{x_i \in V^m} C_i(x_1, x_1', X_2, X_2', \ldots, X_m) = 1$. Since our DBNs are time-variant, in general $C_i$ will be different from $C_i'$ if $t \neq t'$.

A (global) state of the DBN at $t$ will be a member of $V^m$, say $x = (x_1, x_2, \ldots, x_m)$ specifying that $X_i = x_i$ for $1 \leq i \leq n$. This in turn stands for $X_i = x_i$ for $1 \leq i \leq n$ at $t$. Suppose $Pa(X_1') = \{X_1', X_2', \ldots, X_m'\}$. Then a CPT entry of the form $C_i(x_1, x_1', X_2, X_2', \ldots, X_m)$ is $p$ that says if the system is in a state at $t - 1$ in which $X_{ij} = x_{ij}$ for $1 \leq i \leq m$, then the probability of $X_i = x_i$ being the case at $t$ is $p$. In this sense the CPTs specify the probabilistic dynamics locally.

The regular structure of our DBNs induces the function $PA$ given by: $X_t \in PA(X_1)$ iff $X_t \in Pa(X_1')$ for $t = 1, 2, \ldots, T$. We define $i = \{j \mid X_j \in Pa(X_1')\}$ to capture $Pa$ in terms of the corresponding indices.

The probability distribution $Pr(X_1', X_2', \ldots, X_m')$ describes the possible states of the system at time $t$. In other words, $Pr(X_1' = x_1) = \prod_i x_i C_i(x_1)$, one would like to compute $Pr(X_1', X_2', \ldots, X_m')$ for a given $t$.

Fig. 4. The FF algorithm.
We can use the CPTs to inductively compute this:

$$Pr(X^t = x) = \sum_u \left( \prod_i C_i'(x_i \mid u_i) \right) Pr(X^{t-1} = u)$$

(1)

with $u$ ranging over $V^n$.

Since $|V| = K$, the number of possible states at $t$ is $K^n$ and hence explicitly computing and maintaining the probability distributions is feasible only for special cases. One must instead use approximate methods. Here we shall focus on a simple and efficient approximate algorithm called the Factored Frontier (FF) algorithm (Murphy and Weiss, 2001). The key idea of FF is to maintain and propagate joint probability distributions $Pr(X_1^n, X_2^n, \ldots, X_k^n)$ in terms of marginal distributions $\{M_i^t\}$.

The marginal distribution $M_i^t$ is map $M_i^t : V \rightarrow [0,1]$ satisfying $\sum_{v \in V} M_i^t(v) = 1$. Intuitively, $M_i^t(v)$ is the probability of $X_i$ assuming the value $v$ at time $t$. It is given by: $M_i^t(v) = \sum_{x \in X_i^{-1}} Pr(X_i^t = x(j) \mid 1 \leq j \leq n)$.

To describe FF compactly we will assume the following notations. $X_j$ will denote a vector of values over the index set $J \subseteq \{1,2,\ldots,n\}$. It will be viewed as a map $x_j : J \rightarrow V$. We will often denote $x_j(i)$ as $x_i$, if $J$ is clear from the context. Further, we denote by $X$ the vector of random variables $(X_1, X_2, \ldots, X_k)$.

We assume that we are given $M_i^t$ for each $t$. In our application this will correspond to the initial distribution over the intervals of values of the variables in our discretization procedure.

Consequently, if $X_j$ corresponds to an unknown rate constant then $M_i^t(I) = \frac{1}{n}$ in case the values this rate constant can have been partitioned into $n$ uniform sized intervals. To highlight the approximate nature of FF we use $B_i^t$ to denote the marginal probabilities computed by FF and reserve $M_i^t$ for the actual marginal. Using the family $\{M_i^t\}$, FF inductively and approximately computes the marginal probabilities $B_i^t$ using the CPTs as follows.

- $B_i^0 = M_i^0$
- $B_i^t(u) = \sum_{v \in V_i} \left( C_i^t(u \mid v) \prod_{j \in I_i} B_j^{t-1}(v_j) \right)$.

Thus FF generates in one sweep the sequence of (approximate) marginal distribution vectors $(B_1^t, B_2^t, \ldots, B_k^t)$ as illustrated in Figure 4 (for convenience we have assumed that all the rate constants are known). It is an approximation of the sequence of (exact) marginal distribution vectors $(M_1^t, M_2^t, \ldots, M_k^t)$ which in turn will be used to interpret our temporal logic formulas later.

The time complexity of FF is $O(T \cdot n \cdot K^{d+1})$ where $|V| = K$ and $d$ is the maximum over the number of parents that a node can have. Usually $d$ will be much smaller than $n$ and in this sense FF is efficient since its time complexity is linear in $n$.

### 4.2 Probabilistic bounded LTL

In our temporal logic, the atomic formulas (i.e. propositions) will be of the form $(i,v) \# r$ with $\# \in \{\leq, \geq\}$ and $r \in [0,1]$. The proposition $(i,v) \geq r$, if asserted at time point $t$, says that $M_i^t(v) \geq r$; similarly for $(i,v) \leq r$.

The formulas of our logic termed PBL (probabilistic bounded LTL) is then given by: (i) Every proposition is a formula. (ii) If $\phi$ and $\psi$ are formulas then so are $\phi \land \psi$ and $\phi \lor \psi$. (iii) If $\phi$ and $\phi'$ are formulas then so are $O(\phi)$ and $\phi U \phi'$.

The derived propositional connectives such as $\land, \lor, \equiv$ etc. are defined in the standard fashion. The temporal connectives $F$ ("some-time from now") and $G$ ("always from now") are defined in the usual way via: $F(\phi) = true U \phi$ and $G(\phi) = \neg F(\neg \phi)$.

The formulas are interpreted over the sequence of marginal probability distribution vectors $\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_T$ generated by the DBN $\mathcal{D}$. In other words, for $0 \leq t \leq T$, $\mathbf{s}_t = (M_1^t, M_2^t, \ldots, M_k^t)$. Consequently $s_i(t) = M_i^t$ for $1 \leq i \leq n$. We also let $s_i(t) = s_i$ for $0 \leq t \leq T$. We now define the notion of $\sigma(t) = \phi$ ($\phi$ holds at $t$ in $\mathcal{D}$) inductively:

- $\sigma(t) = (i,v) \geq r$ iff $M_i^t(v) \geq r$. Similarly $\sigma(t) = (i,v) \leq r$ iff $M_i^t(v) \leq r$.
- The propositional connectives $\sim$ and $\lor$ are interpreted in the usual way.
- $\sigma(t) = O(\phi)$ iff $\sigma(t+1) = \phi$.
- $\sigma(t) = \phi U \phi'$ iff there exists $t \leq t' \leq T$ such that $\sigma(t') = \phi'$ and for every $t''$ with $t \leq t'' < t'$, $\sigma(t'') = \phi$.

We say that the DBN $\mathcal{D}$ meets the specification $\phi$ and this is denoted as $\mathcal{D} \models \phi$ if $\sigma(0) = \phi$. The model checking problem is, given $\mathcal{D}$ and $\phi$, to determine whether or not $\mathcal{D} \models \phi$.

We begin by letting $SF(\phi)$ denote the set of sub-formulas of $\phi$ and define it as follows. Since $\phi$ will remain fixed we will write below $SF$ instead of $SF(\phi)$.

$SF$ is the least set of formulas containing $\phi$ such that (i) $\phi \in SF$ implies $\phi' \in SF$ (ii) $\phi' \land \phi'' \in SF$ implies $\phi' \lor \phi'' \in SF$ (iii) $\phi' \in SF$ implies $\phi' \in SF$ (iv) $\phi U \phi' \in SF$ implies $\phi' \in SF$.

The main step is to construct a labeling function $st$ which assigns to each formula $\phi' \in SF$ a subset of $\{s_0, s_1, \ldots, s_T\}$ denoted $st(\phi')$. After the labeling process is complete, we declare $\mathcal{D} \models \phi$ just in case $s_0 \in st(\phi)$. Starting with the atomic propositions, the labeling algorithm goes through members of $SF$ in ascending order in terms of their structural complexity. This $\phi'$ will be treated before $\sim \phi'$ is treated and both $\phi'$ and $\phi''$ will be treated before $\phi U \phi''$ is treated and so on.

Let $\phi' \in SF(\phi)$. Then:

- If $\phi' = A$ then $s_i \in st(A)$ iff $\sigma(t) = A$. We run FF to determine this. In other words, $s_i \in st(A)$ iff $B_i^t(v) \geq r$ where $A = (i,v) \geq r$ and $B_i^t$ is the marginal distribution of $X_i$ computed by FF. Similarly $s_i \in st(A)$ iff $B_i^t(v) \leq r$ in case $A = (i,v) \leq r$.
- If $\phi' = \sim \phi''$ then $s_i \in st(\phi')$ iff $s_i \notin st(\phi'')$.
- If $\phi' = \phi_1 \lor \phi_2$ then $s_i \in st(\phi')$ iff $s_i \in st(\phi_1)$ or $s_i \in st(\phi_2)$. 

![Fig. 5. (a) The model (sequence of states) defined by the DBN. (b) The model checking procedure.](http://bioinformatics.oxfordjournals.org/Downloaded from http://biostatistics.oxfordjournals.org)
Suppose $\varphi' = O(\varphi'')$. Then $s_T \not\in st(\varphi')$. Further, for $0 \leq t < T$, $s_t \in st(\varphi')$ iff $s_{t+1} \in st(\varphi'')$.

Suppose $\varphi' = \varphi_1 U \varphi_2$. Then we decide whether or not $s_t \in st(\varphi')$ by starting with $t = T$ and then treating decreasing values of $t$. Firstly $s_T \in st(\varphi')$ iff $s_T \in st(\varphi_2)$. Next suppose $t < T$ and we have already decided whether or not $s_t \in st(\varphi')$ for $t < t' \leq T$. Then $s_t \in st(\varphi')$ iff $s_t \in st(\varphi_2)$ or $s_t \in st(\varphi_1)$ and $s_{t+1} \in st(\varphi')$.

$\varphi' = F(\varphi'')$ and $\varphi' = G(\varphi'')$ can be handled directly. As in the case of $U$, we start with $t = T$ and consider decreasing values of $t$:

- Suppose $\varphi' = F(\varphi'')$. Then $s_T \in st(\varphi')$ iff $s_T \in st(\varphi'')$. For $t < T$, $s_t \in st(\varphi')$ iff $s_t \in st(\varphi'')$ or $s_{t+1} \in st(\varphi')$.
- Suppose $\varphi' = G(\varphi'')$. Then $s_T \in st(\varphi')$ iff $s_T \in st(\varphi'')$. For $t < T$, $s_t \in st(\varphi')$ iff $s_t \in st(\varphi'')$ and $s_{t+1} \in st(\varphi')$.

Due to the fact the model checking procedure just needs to treat one finite sequence as a model, it is particularly simple. Its time complexity is linear in the size of the formula $\varphi$ whereas in traditional settings it will be exponential in the size of $\varphi$.

Figure 5 summarizes our model checking procedure. Properties of pathway dynamics are formulated as PBL formulas. They are then verified using the above labeling algorithm which will call the FF algorithm when dealing the atomic propositions.

5 RESULTS

Here we present our results concerning the GPU implementation and the probabilistic model checking method.

5.1 Performance of the GPU implementation

We have implemented the DBN approximation algorithm on the NVIDIA Tesla 2.0 (Fermi) platform consisting of 4 GPUs of 2 GB global memory each. The 4 units are attached in pairs to two Xeon E5405 @ 2GHz hosts with 16 GB of memory each and the hosts can communicate with each other. We compared the performance of our GPU algorithm with that of a MPI-based C implementation on a cluster of 10 Xeon E5430 @ 2.66 GHz CPUs each with 40 GB of memory. We did so by constructing the DBN approximations for three biopathways models described below. Although the parameter values for these models are known, in each case we set a subset of the parameters as 'unknown' to mimic realistic biopathways models. This considerably increases the computational demands placed on the approximation algorithm. In what follows we briefly describe each of the pathways. The reaction schemes and the corresponding ODE systems can be obtained via the links to the Biomodels database (Le Novere et al., 2006) provided in the Supplementary Information.

5.1.1 The EGF-NGF signaling pathway

PC12 cells proliferate in response to epidermal growth factor (EGF) stimulation but differentiate into sympathetic neurons in response to nerve growth factor (NGF). Brown et al. (2004) developed an ODE model of this pathway. The corresponding reaction network is shown in Figure 6. The model consists of 32 differential equations and 48 kinetic parameters. 20 of the 48 parameters were singled out to be unknown. The ranges of each variable and unknown parameter were discretized into 5 intervals of equal size. The time step $\Delta t$ was fixed to be 6 seconds and $3 \times 10^6$ trajectories were generated up to 600 seconds to fill up the CPTs associated with the DBN approximation.

5.1.2 The segmentation clock network

During the development of vertebrate embryos, the somites are rhythmically produced to establish the segmental pattern of the spines. The periodic formation of somites is driven by the oscillatory expression of a large number of genes. The expression of these genes is controlled by an underlying signaling network called the segmentation clock network (Goldbeter and Pourquie, 2008). The corresponding ODE model consists of 22 differential equations and 75 kinetic parameters. 40 of the 75 parameters were singled out to be unknown. The ranges of each variable and unknown parameter were discretized into 5 equal-size intervals. The time step $\Delta t$ was fixed to be 5 minutes while $3 \times 10^6$ trajectories were generated up to 500 mins to fill up the CPTs.

5.1.3 The thrombin-dependent MLC phosphorylation pathway

The endothelial cells form a dynamic barrier between blood and tissues, which plays an important role in various physiological and pathological processes. The barrier function is determined by the contraction of endothelial cells, which is triggered by the MLC phosphorylation and thrombin is an agonist that can induce the MLC phosphorylation through two different signaling cascades (Maeda et al., 2006). This rather large model consists of 105 differential equations, 110 reactions, and 197 kinetic parameters. In constructing the DBN approximation, we singled out 164 of the 197 parameters to be unknown. We discretized the ranges of each variable and unknown parameter into 5 equal-size intervals and fixed the time step $\Delta t$ to be 2 seconds. To fill up the CPTs, we generated $3 \times 10^6$ trajectories up to 200 seconds.

More details concerning their DBN approximations can be found in the Supplementary Information.

5.2 Performance

The overall runtime are summarized in Table 1. It shows our implementation achieves significant speedup compared to a 10-CPU cluster implementation for the first two case studies. The third case study illustrates the scalability of the GPU implementation. It

Fig. 6. The reaction network diagram of the EGF-NGF pathway (Brown et al., 2004)
5.3 Verification results

For the three case studies we formulated some properties and verified whether they were true or not. For convenience we fixed the values of rate constants and the initial concentrations according to the truth values of the propositions at time 0.

5.3.1 The EGF-NGF signaling pathway

- It is known that the concentration of EGF and NGF remains constantly high. We formulated this property as the formula:

\[ G((\text{EGF}, I_4) > 0.9) \land G((\text{NGF}, I_4) > 0.9) \]

The property was verified to be true.

- The profile of activated ERK is expected to reach a peak after which the concentration begins to fall. The corresponding formula was:

\[ (((\text{ERK}^*, I_0) > 0.6) \land \neg F(((\text{ERK}^*, I_0) > 0.6) \land F(G((\text{ERK}^*, I_0) > 0.6))) \]

The above query was verified to be true.

- We next checked whether the concentration of activated C3G reaches a steady state as experimentally observed. The corresponding formula is:

\[ ((C3G^*, I_0) > 0.8) \land F(G((C3G^*, I_0) > 0.8)) \]

It was verified to be true.

5.3.2 The segmentation clock network

We checked the oscillatory behavior of various species. Following Donaldson and Gilbert (2008), we formulated the property for the oscillatory behavior of Axin as:

\[ F(((\text{Axin}, I_0) > 0.6) \land F(((\text{Axin}, I_2) > 0.6) \land F(((\text{Axin}, I_0) > 0.6) \land F(((\text{Axin}, I_2) > 0.6))) \]

The property specifies the number of peaks and troughs to be expected in an oscillation cycle within the given time bound of the system. Specifically, it says that initially (with a high probability) the system is at the discretized interval 0 followed by a state some time in future where (with a high probability) the system moves to a higher discretized interval and then falls back to initial levels and so on. This query was verified to be true.

5.3.3 The thrombin-dependent MLC phosphorylation pathway

The following are some of the formulas considered for this model:

- The profile of activated Rho starts at a very low level, reaches a high value after which the concentration drops back to the initial level. The corresponding formula was:

\[ ((\text{Rho}^*, I_0) > 0.8) \land \neg F((((\text{Rho}^*, I_0) > 0.8) \land F(((\text{Rho}^*, I_0) > 0.8))) \]

It was verified to be true.

- Rho gets activated and reaches its peak earlier than MLC:

\[ ((\text{MLC}^*, I_0) < 0.1) \land ((\text{Rho}^*, I_0) > 0.8) \land O(F((\text{MLC}^*, I_4) > 0.7)) \]

This was also verified to be true.

- Experimental observations suggest that the concentration of phosphorylated MLC starts at a low value, reaches a high steady state value. The PBL formula used to capture this property was:

\[ ((\text{MLC}^*, I_0) > 0.7) \land F(G((\text{MLC}^*, I_4) > 0.7)) \]

It was verified to be false.

- We then formulated a PBL formula to describe the behavior where the concentration starts with a low value, reaches a high value (peak) after which it drops back to the initial level.

\[ ((\text{MLC}^*, I_0) > 0.7) \land F((((\text{MLC}^*, I_4) > 0.7) \land F((\text{MLC}^*, I_0) > 0.7))) \]

This formula evaluated to be true. This means the current ODE model is unable to explain the experimental data available for this pathway. Further investigation to identify the missing links of the pathway may be required.

6 CONCLUSION

Approximating the ODE based biopathway dynamics as a dynamic Bayesian network allows model analysis tasks such as parameter estimation, sensitivity analysis and probabilistic verification to be efficiently carried out. In this paper we have presented a GPU based implementation for constructing the DBN approximations.

The significant read-sharing in the algorithm will prevent a naive implementation from scaling up to large biopathways models. To overcome this we have proposed a compilation strategy in which heterogeneous threads consisting of trajectory-computing threads and global memory access threads are suitably folded into warps. Further, load balancing was achieved using a simple timing model.

In our experiments we were able to achieve significant speedup compared to a 10-CPU cluster implementation. Furthermore, our method scales well while the cluster based implementation begins to consume infeasible amounts of resources for large models.
We have also formulated a simple probabilistic temporal logic and constructed an approximate but efficient model checking procedure. Though probability enters the picture solely via atomic propositions, one can still formulate many interesting dynamic properties of pathway models. Further, due the fact that there is a single finite run, the model checking procedure is particularly simple. Admittedly it is an approximate procedure. However one can begin with our method to get a preliminary feel for the dynamics and in case a biologically crucial and testable property shows up, one can compute its truth value more precisely by using the HFF algorithm (Palaniappan et al., 2011) (The Supplementary Information contains more details on this).

Both contributions of this paper have wider applicability. In our GPU implementation, we show how data sharing and many accesses to global memory can be tackled. In Monte Carlo based analysis, one must stochastically generate trajectories and check whether they pass a statistical test. This will however entail storing a good deal of information generated by the individual trajectories and multiple passes through this information in cases where the statistical test is based on a temporal property (Donaldson and Gilbert, 2008; Fages and Rizk, 2007). In these settings our mapping techniques will lead to powerful GPU implementations. We plan to demonstrate this in our future work.

The model checking procedure we propose can be applied in other setting where DBNs arise as dynamic models (Sun and Hong, 2007). Further the generic idea of restricting probabilities to just propositions is promising. This will enable classical model checking procedures to be combined with algorithms such as FF and HFF. An interesting challenge will be to develop GPU based implementation for tasks such as parameter estimation and sensitivity analysis that are computationally demanding but appear to offer opportunities for a parallel implementation. In this context, the sum-of-product algorithm implementation presented in Silberstein et al. (2008) promises to offer helpful pointers.

REFERENCES


