# Adaptive Tree-Based Search for Stochastic Simulation Algorithm


## Vo Hong Thanh

University of Trento, Italy
E-mail: vo@disi.unitn.it

## Roberto Zunino

University of Trento and COSBI, Italy
E-mail: roberto.zunino@unitn.it



**Abstract:** Stochastic modelling and simulation is a well-known approach to predicting the behaviour of biochemical systems. Its main applications lie in those systems in which the inherently random fluctuations of some chemical species are significant, as often is the case whenever just a few macromolecules can have a large effect on the rest of the system. A better understanding of these phenomena is important for the towards development of effective therapeutic treatments. The stochastic simulation algorithm (SSA), which was introduced by Gillespie, is a standard method to properly realize the stochastic nature of reactions. A large effort is then ongoing to improve the performance of SSA so to make it possible to simulate larger models. In this paper we propose an improvement to the SSA based on the Huffman tree, a binary tree which is used to define an optimal data compression algorithm. We exploit results from that area to devise an efficient search for chemical reactions to be fired, moving from linear time complexity to logarithmic complexity. We combine this idea with others from the literature, and compare the performance of our algorithm with previous ones. Our experiments show that our algorithm is faster, especially on large models.



**Keywords:** Systems Biology ; Biological simulation; SSA ; Tree Search SSA ; Huffman Tree SSA


## 1 Introduction

Modelling and simulation play an important role in the development of computational systems biology. A model allows to encode the reaction network in a precise mathematical form, and its dynamical behaviour can be visualized by conducting *in silico* simulation. The models which are validated can be applied to discover new relations in the systems e.g., see Purutcuoglu and Wit (2006). Different modelling techniques have been introduced to cope with the complexity of bichemical reaction systems. The deterministic approach models a biochemical reaction system as a collection of ordinary differential equations (ODEs), which often derives from the *law of mass action*. The mass-action kinetics

postulates the changes in population of species by reaction firings have a negligible effect to the macroscopic trend of the molecular concentrations. The population of species thus approximates continuous. However, the population number of species in a reaction network is obviously discrete. Furthermore, it is common in a model to find a few specific species, e.g., gene, RNA, which play a key role, yet have a small population. A random reaction between these species could lead to a significant quantitative and qualitative fluctuation in the system behaviour (see McAdams and Arkin (1999, 1997); Raser and O'Shea (2005); Pedraza and van Oudenaarden (2005) for detailed discussions). The stochastic approach recently has been emerged as an alternative for the deterministic approach due to its ability to capture the inherently dynamic and discrete nature of the biochemical systems.

The stochastic kinetics is grounded on the probability of a reaction firing can be expressed by a *propensity function*. Gillespie (1992) provided a rigorous derivation and justification of stochastic modeling of biochemical reactions in form of the chemical master equation (CME). However, CME is hard to solve analytically or numerically, unless the system is very small. In spite of that, its solutions can be realized by a simulation procedure called the stochastic simulation algorithm (SSA), which is an exact simulation method for reproducing stochastic noise in biological reactions.

Basically, SSA uses a Monte Carlo simulation technique to sample the system state, and makes the system evolve by firing one reaction at a time. There are two well-known implementations of the SSA method, namely the First Reaction Method (FRM) and the Direct Method (DM) (see Gillespie (1976, 1977)). DM and FRM are mathematically equivalent but differ in how to compute the next reaction firing. In FRM, the putative times of all reactions are generated, after which the smallest one is chosen to fire. In contrast, DM generates the next firing time, and then a search is conducted to find the reaction to fire. The system is updated accordingly after that, and the algorithm proceeds with the next simulation step.

SSA, however, is very computational demand for simulating large models which are often needed to perform a thorough analysis. Thus, several efficient formulations have been introduced to both FRM and DM to make it capable for large reaction networks. The Next Reaction Method (NRM) developed by Gibson and Bruck (2000) improves FRM by representing the dependencies among reactions using a graph, and employing a special indexed structure, a priority queue in this particular situation, to store putative times. Hence, less time is spent for choosing a reaction and updating the system state. NRM is often faster than FRM and DM, but it exposes a challenge for implementing an efficient priority queue. The Optimized Direct Method (ODM) by Cao et al. (2004) and Sorting Direct Method (SDM) by McCollum et al. (2006) improve DM based on a careful observation that the search will be faster when reactions are indexed according to their frequencies. Variants of SSA by dividing reactions into groups have been proposed. Mauch and Stalzer (2011) proposed a 2D search for finding the next reaction in which the propensities are stored in a matrix. The next reaction firing is found by two linear searches: one finds the row using partial sums, and the other finds the reaction within the row. The grouping of reactions is also exploited by Schulze (2008); Slepoy et al (2008), but the search of the next reaction in group now is discovered by an acceptance-rejection procedure. This formulation is referred to as the composition rejection SSA (CR-SSA). The time complexity for the long run of CR-SSA is constant time; however, in order to achieve the constant time complexity CR-SSA postulates two assumptions: 1) the number of affected reactions of a reaction firing should be a constant factor, and 2) the reaction propensities after updated by a reaction firing should vary less significantly. The CR-SSA performance can be very slow once these assumptions

are violated. This has been shown experimentally by Mauch and Stalzer (2011). Instead of grouping reactions, reactants are grouped (see Ramaswamy et al. (2009); Indurkhya and Beal (2010)). For example, the partial propensity SSA (PSSA) developed by Ramaswamy et al. (2009) computes only the "partial" propensities related to a reactant. A complex data structure is exploited to store reactants as well as partial propensities. An equivalent procedure with SSA is used to find the next reaction firing. PSSA will outperform over SSA for highly coupled reaction models because the search time complexity of PSSA is proportional with the number of species. However, this method only supports for reactions which are in the elementary reaction form, i.e., having at most two reactants, and use the mass-action kinetics (see Ramaswamy et al. (2009)). PSSA therefore requires more effort to simulate models that contain abstraction in reactions and/or complex kinetics such as the enzymatic reaction with Michaelis–Menten kinetics (see Crampina et al. (2004)). The parallel computing environment also has been exploited recently to reduce the simulation time of SSA. There are two main types of parallel techniques used: data parallelism and functional parallelism (e.g., Li and Petzold (2010); Dittamo and Cangelosi (2009); Thanh and Zunino (2011)). Beyond exact methods, approximate methods have been presented to reduce the simulation time. For example, the $\tau$-leaping method (see Gillespie (2007, 2001); Gillespie and Petzold (2003); Cao et al. (2006)) assumes that the propensity of some reaction is roughly constant for a small amount of time, so that the simulation clock can be advanced for that amount in one step.

Simulating large models with DM, using a linear search to determine the reaction to fire may therefore be rather inefficient. In this work, we study the impact of such search on the overall simulation performance, and contribute to its improvement by applying variants of a tree-based search by Thanh and Zunino (2012); Li and Petzold (2006); Blue et al. (1995)). To speed up the propensity update, we also exploit a dependency graph. We show, both in theory and in practice, that by using an underlying tree data structure to store reaction propensities the search time can be sensibly improved. Theory shows that our approach reduces search time from linear to logarithmic, although propensity updates now require logarithmic time instead of constant time. Theory also predicts the shape of the tree leading to optimal average search time. This turns out to be the Huffman tree by Huffman (1952), a device used in computer science for data compression. Experiments confirm that this tree indeed leads to faster simulation. This article is a revised and expanded version of a previous short paper of ours (Thanh and Zunino (2012)). Here, we further study the impact of an adaptive approach, by which the propensity Huffman tree is rebuilt during the simulation depending on how the system evolves.

The paper is organized as follows: in the next section (section 2) we review the main stochastic simulation methods. In section 3, we describe the tree-based approach for reducing the computation cost for finding the next reaction firing. Section 4 gives some experimental results for our approach. The concluding remarks, and some possible future research directions are provided in section 5.

## 2 SSA

We consider a well-stirred biochemical reaction system with $n$ species denoted as $S_1, ...S_n$, which interact through $m$ reactions $R_1, ...R_m$. Each reaction has the following general form:

$$R_j : v_{1j}S_1 + ... + v_{nj}S_n \xrightarrow{k_j} v'_{1j}S_1 + ... + v'_{nj}S_n \quad (1)$$

where $v_{ij}$ and $v'_{ij}$ are the *stoichiometric coefficients*, and $k_j$ is called (stochastic) *rate constant* of reaction $R_j$. In the case the system contains reversible reactions, they can be modelled as two irreversible reactions.

The dynamic state of the system is represented by a vector $X(t) = (X_1(t), ..., X_n(t))$ where $X_i(t)$ denotes the population of species $S_i$ at time $t$. If the next reaction $R_j$ fires at time $t + dt$, then the system state changes by the amount denoted by vector $v_j$ where $i$th element equals to $v'_{ij} - v_{ij}$, which describes the change in the population of species $S_i$. The state transition of the system therefore can be formulated as:

$$X(t + dt) = X(t) + v_j \tag{2}$$

Let $P(x, t)$ be the probability of system being in state $x$ at time $t$. The following differential equation expresses the time evolution of $P(x, t|x_0, t_0)$ given the initial state $X(t_0) = x_0$ at time $t_0$.

$$\frac{\partial P(x, t|x_0, t_0)}{\partial t} = \sum_{j=1}^{m} \left[ a_j(x - v_j)P(x - v_j, t|x_0, t_0) - a_j(x)P(x, t|x_0, t_0) \right] \tag{3}$$

where $a_j$ is the *propensity function*, defined such that $a_j(x)dt$ is the probability that reaction $R_j$ is fired in the next time $t + dt$.

Equ. 3 is generally called the chemical master equation (CME). It is in fact a collection of differential equations which describe all possible state transitions by reactions. CME thus completely determines the time evolution of the system. However, an analytic solution of this equation is hard to find, unless the system is rather simple. Although the CME equation cannot be solved in general, its solutions can be exactly sampled by simulating the next reaction probability function $p(\tau, j|x, t)$, in which $p(\tau, j|x, t)d\tau$ is the probability a reaction will be fired in the next time $t + \tau + d\tau$ and it is $R_j$, provided that we are in state $X(t) = x$. We have:

$$p(\tau, j|x, t) = a_j(x)exp(-a_0(x)\tau) \tag{4}$$

where

$$a_0(x) = \sum_{j=1}^{m} a_j(x) \tag{5}$$

while $\tau$ and $j$ are the time of the reaction firing and its index, respectively.

Based on Equ. 4-5, the Direct Method (DM) computes $\tau$ and $j$ by generating two random numbers, called $r_1$ and $r_2$, from the uniform distribution $U(0, 1)$, and then applies the inversion method to obtain the values of $\tau$ and $j$ as:

$$\tau = \frac{1}{a_0(x)} ln \left( \frac{1}{r_1} \right) \tag{6}$$

$$j = \text{the smallest } j \text{ s.t. } \sum_{k=1}^{j} a_k(x) > r_2 \; a_0(x) \tag{7}$$

| No. | Reaction |
|---|---|
| 1 | A + B ⟶ C |
| 2 | C + B ⟶ D |
| 3 | E + C ⟶ 2C + F |
| 4 | 2C ⟶ G |
| 5 | F + H ⟶ B |

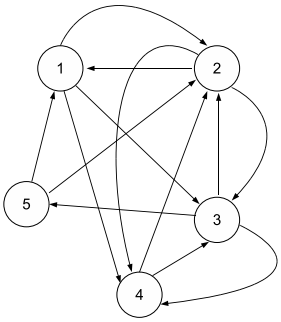


**Figure 1** Dependency graph (removing self affected edges)

A mathematically equivalent procedure is provided by the First Reaction Method (FRM). In the simulation loop, the algorithm generates $m$ random numbers $r_1 \dots r_m$ from the uniform distribution $U(0,1)$, and computes the putative times for each reaction as:

$$\tau_j = \frac{1}{a_j(x)} ln\left(\frac{1}{r_j}\right), j = 1...m \tag{8}$$

The smallest value $\tau_j$ is set as the firing time $\tau$, and the corresponding reaction $R_j$ fires.

Propensity $a_j(x)$s are computed once at the start of the simulation, and then updated as soon as the state $x$ changes. To speed up the updates, it is common to exploit a dependency graph between reactions, which describes which propensities actually need to be recomputed after every reaction firings. The dependency graph $DG(V,E)$ is a directed graph (see Fig. 1 for an example of dependency graph for Oregonator model ) which contains the reactions as vertices $V$, while an directed edge $e(R_i, R_j) \in E$ if and only if $R_j \in$ affects$(R_i)$, the set of reactions affected by $R_i$. Formally

$$\text{affects}(R_i) = \{\ R_j\ |(\text{reactants}(R_i) \cup \text{products}(R_i)) \cap\ \text{reactants}(R_j) \neq \emptyset\} \tag{9}$$

where reactants($R_i$) and products($R_i$) are the set of species taking part in reaction $R_i$ as reactants and products, respectively. Because a direct catalyst is not consumed by the reaction itself, it is excluded from the reactants and products of the reaction.

## 3 Binary Search for SSA

For large models, binary search is a more efficient method than the linear search which is used in DM (logarithmic vs. linear complexity). In order to exploit binary search the partial sums of propensities had to be computed and stored in a tree structure. A dependency graph for updating the system is also incorporated to further improve the simulation performance. In this section we first detail the data structures and algorithms used to apply binary search on complete trees with a dependency graph based update mechanism. Then, we switch from complete trees to Huffman trees, and exploit their optimal data compression property to minimize the number of comparisons needed to find the next reaction firing.

### *3.1 Complete Tree Search*

A (binary) complete tree, is a binary tree completely filled at every level, except possibly the last; each node has exactly two children (internal node), or zero (leaf). For our purposes, leaves hold the reaction propensity $a_j$ for $j = 1 \cdots m$, while internal nodes store the sums of values of their child nodes. Thus, at the top, the root holds the total sum $a_0$. Proposition 1 and the following discussion allow to store a complete tree on a contiguous array, hence improving cache-friendliness.

**Proposition 1:** *A complete binary tree with $m$ leaves has exactly $2m - 1$ nodes.*

We therefore use an array with $2m - 1$ elements to represent a complete tree with $m$ reaction propensities filled at the lowest level. In the array representation, a node at position $i$ will have its two children at position $2i$ and $2i + 1$. We then recursively from leaves to root construct the tree with the internal sums as in Algo. 1. Each element of the array TREE stores only the partial sums of the reaction propensities, so we simply need each cell to store a single value (a floating point double). In order to build up the tree, the number of reactions $m$ must be an even number. In the case $m$ is not one can add a dummy node (with propensity $0$) as the last element of the array.

**Algorithm 1** Building the complete tree

**procedure:** *build_tree*(position)
**require:** array TREE with $2m - 1$ elements where elements from $m$ to $2m - 1$ are filled with reaction propensities

```
1: if position is not leaf then
2:   build_tree(2position)
3:   build_tree(2position + 1)
4:   TREE[position] = TREE[2position] + TREE[2position + 1]
5: end if
```

Once having built the tree, to search for the next reaction firing we proceed as follows. Let $r$ be a random number in $U(0, 1)$, and $ra_0$ be the value we are looking for, as in Equ. (7). Starting from the root, we travel down the tree, following the left or right branches according to whether the propensity sum stored in the left one is smaller than the search value. Whenever we take a right branch, we adjust the search value by subtracting it from the value stored in the parent. The whole procedure is outlined in Algo. 2. The procedure is correct, in the sense it finds the same leaf $R_j$ as in Equ. (7), so each reaction indeed is chosen with the correctly desired probability $a_j/a_0$.

The reaction firing causes the system state change; therefore, we also have to update the propensity tree as well. For that, we use the dependency graph to keep the local affection between reactions and exploit the fact that the parent of node $i$ is located at position $\lfloor i/2 \rfloor$. Hence, we only update the reactions affected and their ancestor nodes in the tree following the path from leaf to root. Since the average path length is $\log(m)$, the total cost for the simulation is stable $O(\log(m))$.

The order of reactions in the leaves of a tree may have an impact on the update performance of the affected reactions. To illustrate the idea, imagine a binary tree containing two reactions affecting each other. If they are stored in sibling leaves, their update can be

**Algorithm 2** Finding the next reaction firing

**procedure:** *search*(position, s)
**require:** properly set up array TREE, search value s
1: **if** position is leaf **then**
2: **return** position
3: **else if** TREE[2position] $\geq$ s **then**
4: search(2position, s)
5: **else**
6: v = TREE[position] - s
7: search(2position + 1, v)
8: **end if**

twice more efficient than updating two reactions having the root as their single common ancestor: this is because the partial sums stored in the tree path can be updated just once. Ideally, groups of reactions which affect each other should be placed as close as possible in the tree.

### *3.2 Huffman Tree Search*

While storing reactions in a complete tree minimizes the *height* of the tree, corresponding to the average computation to search the next reaction firing, this does not lead to an optimal average-case performance. Indeed, consider the average number of comparisons performed during the search of the next reaction firing and denote this value by $T_m(C)$, we have:

$$T_m(C) = \sum_{j=1}^{m} w_j D_j \tag{10}$$

where $m$ is the total number of reactions, $D_j$ is the depth of leaf $j$ in the tree, and $w_j$ is the weight corresponding to the probability the reaction $R_j$ is being selected to fire, respectively. The reaction depth $D_j$ is indeed the search length for firing reaction $R_j$. Optimizing SSA, by our formulation, therefore now is done by finding a tree representation which optimizes $T_m(C)$.

In a complete tree setting, the depths $D_j$ are roughly equal, since all the leaves are in the last level or in the next-to-last one. So, we are performing the same number of computations in every case i.e., the likely event of picking a fast reaction requires the same computational effort of the unlikely event of picking a slow reaction. It is simple to check that this choice leads to a non optimal $T_m(C)$. Consider the extreme case in which reaction $1$ has $91\%$ probability, while reactions $2, 3, 4$ have $3\%$ probability each. In a complete tree, we would have $D_j = 2$, hence $T_4(C) = 2$. With a non-complete tree it would however be possible to move reaction $1$ up in the tree ($D_1 = 1$), while moving the other reactions down ($D_j = 3, j > 1$). This leads to $T_4(C) = 1.18$ comparisons, which is better. Intuitively, we can improve the performance of the complete tree search, especially for multi-scale biochemical systems, which can be separated into fast and slow reactions. The main idea would then be to place fast reactions close to the root, while slow ones farther from it.

These facts are very closely related to well-known results in data compression. Indeed, the minimization of $T_m(C)$, which leads to optimal performance in our setting, is the purpose of the Huffman encoding for data compression. Huffman tree, in Huffman (1952);

Knuth (1968), provides a possible construction to minimize $T_m(C)$. The fundamental idea there is to build the tree by repeatedly merging trees in a forest, which initially contains only trees with one node. At each step, the two trees whose roots ($p$ and $q$) have the *smallest* weights ($w_p$ and $w_q$) are merged. A new root $pq$ is created and the two previous trees become the subtrees of $pq$. The $pq$ node is assigned weight $w_{pq} = w_p + w_q$. This is repeated until the forest contains only one tree. From this, it is clear that in the final tree we have $D_{pq} + 1 = D_q = D_p$, where $p, q, pq$ are the nodes involved in any merge. Hence, we obtain for any such $p, q, pq$:

$$\begin{aligned} T_m(C) &= \sum_{\substack{j=1 \\ j\neq p,q}}^{m} w_j D_j + w_p D_p + w_q D_q \\ &= \big( \sum_{\substack{j=1 \\ j\neq p,q}}^{m} w_j D_j + w_{pq} D_{pq} \big) + w_{pq} \\ &= T_{m-1}(C) + w_{pq} \end{aligned} \tag{11}$$

which relates $T_m(C)$ with $T_{m-1}(C)$. The above allows us to recall the main result for Huffman trees.

**Proposition 2:** *The Huffman tree gives the minimum value of* $T_m(C)$

*Proof*: By induction on $m$. **Base case**: easy to check for $m = 2$. **Inductive case**: by the inductive hypothesis, the Huffman tree for $m - 1$ gives the optimum value for $T_{m-1}(C)$. By contradiction, suppose the Huffman tree for $m$ is not optimal. So there is some tree having total number of comparisons $T'_m(C)$ such that $T'_m(C) < T_m(C)$. W.l.o.g. the smallest weights must be placed at lowest level. Hence, let $p$ and $q$ are nodes with smallest weight and their parent labeled $pq$. Using (11), we have $T'_{m-1}(C) + w_{pq} < T_{m-1}(C) + w_{pq}$ then $T'_{m-1}(C) < T_{m-1}(C)$, contradicting the inductive hypothesis. □

Since each node in Huffman tree has two children, Proposition 1 still holds. We therefore still use an array with size $2m - 1$ for representing the Huffman tree. Note that, however, we do not need $m$ to be even in this setting. The elements at position from $m - 1$ to $2m - 1$ are filled by reactions as leaves. But, unlike for complete trees, each element in the array must point to its left and right child. Building a Huffman tree is done by employing a heap to extract the nodes $p, q$ with minimum weight at each step.

The Huffman tree we built in the Algo. 3 is stored in an array in which each element contains the fields: VALUE, LEFT, RIGHT. The partial sum of propensity is now stored in the VALUE field. The index of left and right subtree is indicated by LEFT and RIGHT, respectively. The same binary search procedure in Algo. 2 is applied to search the Huffman tree for the next reaction, except that now LEFT and RIGHT fields are used to travel the tree, instead of the previous method which works only for complete trees.

The update stage in the simulation is to reflect the changes to the propensity of reactions affected. Each element of array TREE stores the location of its parent node by an additional field, called PARENT, which is set up in the Huffman tree building procedure. The path from a leaf to its root is thus easily traversed. Accompanying with dependency graph, we traverse upward this path to update reactions affected. In the following we discuss about the weight function in the implementation of Huffman tree and the tree rebuilding when the tree becomes non-optimal.

**Algorithm 3** Building Huffman tree

**procedure:** *build_huffman_tree*

**require:** An array TREE with $2m-1$ elements, where the elements from $m$ to $2m-1$ are filled

1: build heap H with elements $(m, w_1)$,... $(2m-1, w_m)$, ordered according to $w_j$
2: **for** $position = m-1$ down to 1 **do**
3: extract top element ($p$, $w_p$) from H
4: extract top element ($q$, $w_q$) from H
5: TREE[$position$].VALUE = TREE[$p$].VALUE + TREE[$q$].VALUE
6: TREE[$position$].LEFT = $p$
7: TREE[$position$].RIGHT = $q$
8: insert($position$, $w_p + w_q$) into H
9: **end for**

By applying the Huffman tree to find the next reaction firing, we want to reduce the number of comparisons of SSA. A natural candidate for the weight function is the propensity function $a_j$ since this choice leads to less time spent for finding the next reaction (which has large propensity). However, during the execution of the simulation, reaction firings affect their dependent propensities, which also could change rapidly. This happens, for example, whenever a reaction has a very large rate constant but a small number of reactant molecules. Its propensity will significantly change by a very large amount. Updating the values stored in the tree therefore could make the tree no longer optimal i.e., no longer an Huffman tree. In this case, we face the choice of either proceeding with a non-optimal tree (which could still be near the optimum, though), or rebuilding the Huffman tree. Rebuilding the tree is rather expensive, so we need a trade-off.

Our idea is to postpone the reconstruction of the tree unless the change of the weight is significant. We thus keep on using a non-optimal tree for some tunable number of SSA steps. Note that the choice of this number of steps only affects performance, while the results are still exact. Several strategies can be applied to this aim, as we detail in the rest of this section.

**Fixed time tree rebuilding.** An intuitive and simple strategy for the reconstruction discussed above is to use a fixed number $k$, and rebuild the tree structure only once every $k$ steps. This amounts to assuming that the weights do not change significantly during $k$ simulation steps, so we can postpone the rebuilding without a large impact on performance. To compensate, we slightly modify weights $w_j$ to cope with propensities changing rapidly. More precisely, we assign a higher weight to those reactions which are more likely to change.

For reaction $R_j$, we consider two sets: conflicts($R_j$) is the collection of reactions that affect and compete with $R_j$

$$\text{conflicts}(R_j) = \{R_i | R_j \in \text{affects}(R_i), \text{reactants}(R_i) \cap \text{reactants}(R_j) \neq \emptyset\} \qquad (12)$$

and favors($R_j$) is the collection of reactions that affect and favor $R_j$

$$\text{favors}(R_j) = \{R_i | R_j \in \text{affects}(R_i), \text{products}(R_i) \cap \text{reactants}(R_j) \neq \emptyset\} \qquad (13)$$

respectively, where reactants($R_j$) and products($R_j$) are the set of species taking part in reaction $R_j$ as reactants and products. In terms of the dependency graph $DG(V, E)$, we have the following relation: |conflicts($R_j$)| + | favors($R_j$)| = in-degree($R_j$).

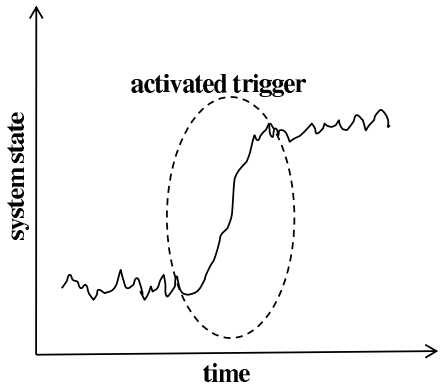


**Figure 2** Random noise activates the trigger causing the system to abruptly exit of its stable state.

Then, when a reaction fires we will estimate the probability it will increase (resp. decrease) the propensity of a reaction $R_j$ as $|\text{conflicts}(R_j)|/m$ (resp. $|\text{favors}(R_j)|/m$). For $k$ simulation steps, the estimated weight of reaction $R_j$ is:

$$w_j(a_j, k) = a_j + \alpha_1 k \frac{|\text{favors}(R_j)|}{m} + \alpha_2 k \frac{|\text{conflicts}(R_j)|}{m} \tag{14}$$

where $\alpha_1, \alpha_2$ are parameters denoting the average change amount. For simplicity, we assign these to the stochastic rate constant for the reaction at hand i.e, $\alpha_1 = -\alpha_2 = k_j$.

**Adaptive time tree rebuilding.** Choosing a good value for $k$ is a model-dependent task. Indeed, using a fixed rebuilding every $k$ reactions seems to be appropriate in simulating systems which are almost stable, so that propensity changes are small. Models which are not stable, but instead can exhibit significant fluctuations, may require a better rebuilding strategy. We therefore improve the above approach by no longer using a fixed number of steps, and instead resorting to an adaptive approach: we rebuild the tree only when a significant change occurred i.e., when rebuilding has more chances to lead to a significant gain in performance.

Large changes of reaction propensity occur, e.g., when the reaction rate constant is very large, hence even a small fluctuation in reactant population can lead to a very different reaction propensity. Large changes to propensity may also happen for reactions having a medium rate constant when the population of reactants suddenly is increased by other fast reactions. These types of biochemical reactions are typically found in e.g., *biochemical switches*. There, the system spends a bulk of time fluctuating near the stable state; however, when random noise triggers the switch this results in a dramatic change in the propensities of reactions (shown in Fig. 2). In that case, the tree has to be rebuilt, or the search performance will suffer.

A straightforward procedure to predict such events is based on trial simulations, in which sample trajectories are collected. An analysis on the ensemble of trajectories is then performed to obtain roughly average times for the fluctuations in the system. These values therefore are used as the times to rebuild the Huffman tree. However, since the behavior of biochemical systems is inherently random, there will always be some difference between the predicted times and real simulation.

An intuitively less expensive approach is to dynamically check for the changes in propensities caused by each simulation step. The advantage of this approach is that we

**Table 1** Models with number of reactions and species

| Model | Species | Reactions |
|---|---|---|
| Oregonator | 8 | 5 |
| Circadian Cycle | 9 | 11 |
| HSR of E. Coli | 28 | 61 |
| MAP Kinase Cascade | 106 | 296 |

detect the abrupt changes on the fly. To do so, we define an acceptance threshold $\delta$, which is the largest change which does not trigger an immediate tree rebuilding. Let $\tau$ be the sojourn time until the next reaction fires. Then, the difference in propensity of a reaction $R_j$ is computed as:

$$c_j(\tau) = a_j(x(t+\tau)) - a_j(x(t)) \tag{15}$$

where $t$ is the current simulation time. When the above difference is high enough, i.e., $c_j(\tau) \geq \delta$, we then immediately restructure the Huffman tree.

In the discussion above we only consider abrupt single changes to propensities. A different approach is to also account for the fact that small updates, when applied many times, can also cause a significant change in propensities. To handle this case as well, we cumulatively sum all the propensity updates since the last tree rebuilding while simulating, as sketched below.

$$s_j \mathrel{+}= \sum_{\tau} c_j(\tau) \tag{16}$$

Thus, we rebuild the entire tree when the cumulative sum is over the acceptance threshold i.e., when $s_j \geq \delta$.

## 4 Experimental Results

### *4.1 Evaluation of fixed time tree rebuilding*

In this section, we report on simulation results for several models differing in size. Table 1 provides a summary of the number of reactions and species in each model. In the following we give a brief description of the models.

The first two models we studied are the Oregonator and Circadian Cycle model. The underlying mechanism of the Oregonator dynamics contains both an autocatalytic step and a delayed negative feedback loop. It is a kind of chemical reaction that shows a periodic change in the concentrations of the products and reactants Alonso et al. (2006). The second model is the simplified circadian cycle model in Vilar et al. (2002). The circadian rhythm is a daily cycle in the biochemical processes of many living beings. The key mechanism of the circadian rhythm is the intracellular transcription regulation of two genes, that is, an activator and a repressor. Activator acts as the positive element in transcription in binding to promoter, while repressor acts as the negative element by repressing the activator.

The third model that we simulated is the heat shock response (HSR) process which occurs when cells are shifted to high temperature. The synthesis of a small number of proteins, called the heat shock proteins (HSPs), is rapidly induced. In E. coli, the response

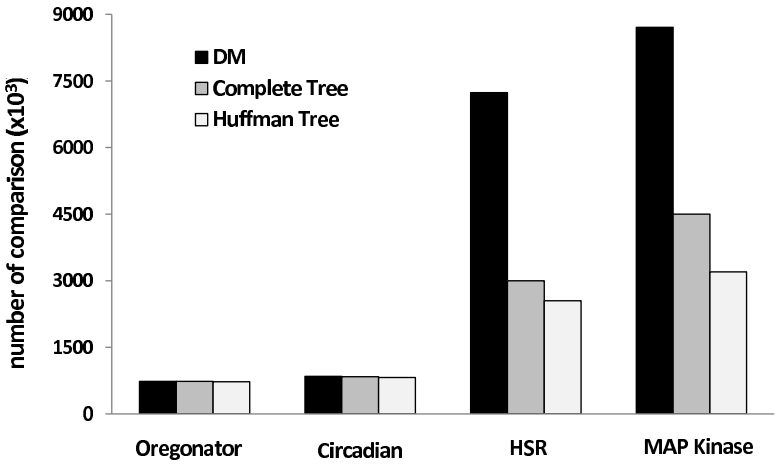


**Figure 3** Number of comparisons performed by each algorithm. For Huffman Tree, we rebuild the tree every $k = 100,000$ steps.

is controlled by the so-called $\sigma$-factor which is capable of binding to various regions of the DNA that stimulate the transcription of the particular gene under their control. When E. coli senses the raised temperature the special heat shock $\sigma$-factor called $\sigma 32$ will replace $\sigma 70$, which is the bound $\sigma$ unit of RNA Polymerase (RNAP), to accelerate HSPs synthesis (see more detail in Kurata et al. (2001)).

The last model is the the mitogen-activated protein (MAP) kinase (MAPK) cascade. The MAP kinase signaling pathway is a chain of proteins in the cell that cascade a signal from a receptor on the surface of the cell to its nucleus. The signal begins when mitogens or growth factors bind to the receptor on the cell surface and ends when the cell produces a response pattern e.g., growth, differentiation, inflammation and apoptosis. The cascade is well-conserved which means this process can be found in a large number of cell types. The basic mechanism of this pathway is driven by three protein kinases: MAPKKK (such as RAS/Raf), MAPKK (such as MEK) and MAPK. The external stimuli activate the first element of the pathway, the MAPKKK. The activated MAPKKK phosphorylates MAPKK at two sites. The phosphorylated MAPKK then activates the MAPK through the phosphorylation on its threonine or tyrosine of the protein structure. MAPK can then act as a kinase for transcription factors, but may also have a feedback effect on the activity of kinases like the MAPKKK further upstream Kolch (2000).

The performance of our algorithm is reported in Fig. 3-4. Four algorithms are compared: DM, NRM, Complete Tree Search, Huffman Tree Search. The results have been computed for $500,000$ simulation steps on an Intel Core i5-540M processor. The DM algorithm we used was adapted so to exploit a reaction dependency graph in updating the propensities of the affected reactions. For Huffman Tree Search, we applied the fixed time tree rebuilding, so we had to pick a number $k$ of steps after which we reconstruct the Huffman tree. In this experiment we chose $k = 100,000$, hence causing the tree to be rebuilt $5$ times in the whole simulation.

In Fig. 3 we show the number of comparisons performed for finding the next reaction firing in each case. The NRM algorithm is not shown because the smallest putative time is always on the top of the priority queue used in NRM. In all the cases, the Huffman tree search performed the least number of comparisons. When simulating small models,

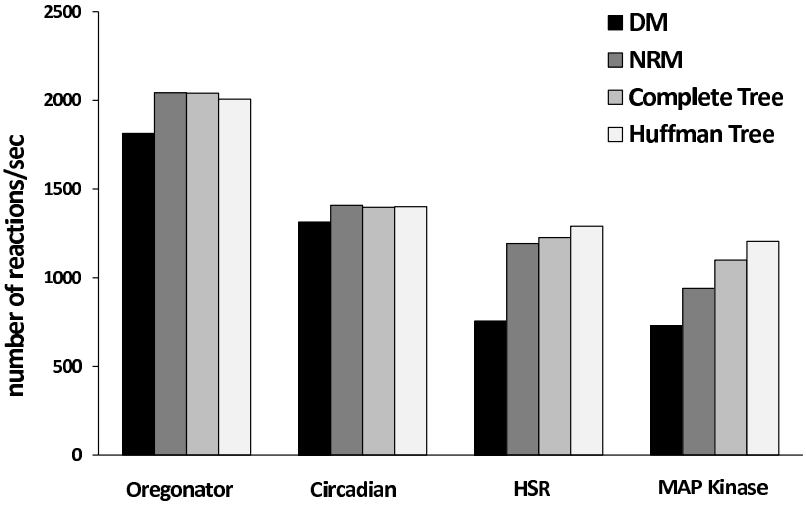


**Figure 4** Overall performance in terms of reactions fired per second.

the difference between linear search and binary search is not very significant. However, when using the larger models binary search is nearly $50\%$ faster than linear search, and Huffman Tree Search further improves on that by performing $\sim 20\%$ fewer comparisons than Complete Tree Search.

In Fig. 4 the overall performance of the different algorithms is compared. The simulation of small models is not significantly affected by the choice of the algorithm. This is intuitive, since in these models there is little room to improve both in search time and in update time, which contribute roughly in the same way to the overall performance. However, when the system is large, then search time dominates update time. In this case, search time significantly benefits from using an algorithm such as Huffman tree search, as our results for the MAP Kinase model show.

Picking an appropriate $k$ to gain the best performance depends on the problems at hand. More specifically, it depends on the changes of the propensity function. In general, we could pick a large value $k$ for systems which evolve near a stable state, so that changes in propensity are small. For unstable systems where the propensities sharply change frequently, by contrast, the value of $k$ should be chosen small enough to capture such fluctuation. In practice, one can roughly estimate the value of $k$ from a pilot simulation run, or move to an adaptive approach for tree rebuilding.

### *4.2 Evaluation of adaptive time tree rebuilding*

In this section, we compare the performance of the Huffman tree search with fixed and adaptive time rebuilding. To this aim, we considered the gene regulation given in Table 2. There, a single gene is being translated into mRNA, which is then being transcribed into proteins. While there is no transcription factor binding to DNA, the transcription occurs at a medium rate. The system then slowly fluctuates for a long time. However, as soon as the transcription factor IND_Protein binds to DNA, it acts as an inducer, causing the transcription to happen at a larger rate by quickly producing a large amount of mRNA. In Fig. 5 we report the simulation runtime of these approaches. We measured the average time

**Table 2** Gene expression model

| |
|---|
| DNA $\xrightarrow{k_1=5}$ DNA + mRNA |
| mRNA $\xrightarrow{k_2=5}$ mRNA + Protein |
| mRNA $\xrightarrow{k_3=1}$ $\emptyset$ |
| Protein $\xrightarrow{k_4=1}$ $\emptyset$ |
| DNA + IND_Protein $\xrightarrow{k_5=0.0001}$ DNA_INDProtein |
| DNA_INDProtein $\xrightarrow{k_6=100}$ DNA_INDProtein + mRNA |

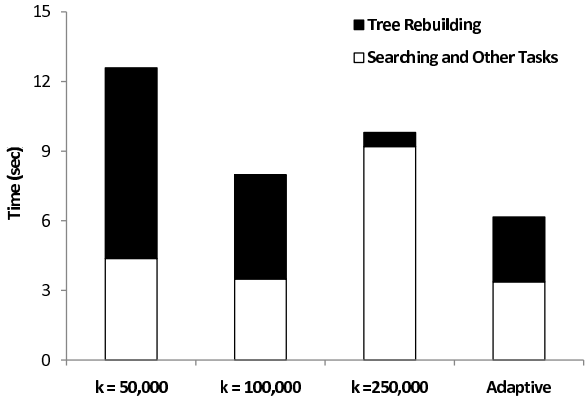


**Figure 5** Simulation time of fixed time and adaptive time effort

required to run a simulation for $500,000$ steps (disregarding the initial setup time for the algorithms).

In our gene expression model, the inducer protein IND_Protein has an important role while binding to DNA, since it accelerates the rate of mRNA production, which results in a large amount of protein. This is because the last reaction in the model has very large rate $k_6 \gg k_1$, while its reactant population is small. The Huffman tree structure for applying binary search clearly should be consolidated when this reaction occurs.

The adaptive approach performs the reconfiguration at the correct time, by dynamically checking for propensity changes. Even if this requires a small overhead, still this leads to a better performance than those we obtained through the fixed approach, as Fig. 5 shows. By contrast, in the fixed approach, a small value of parameter $k$ causes the tree to be rebuilt too many times. Here, although the tree is kept near the optimum, and less time is spent for searching, the rebuilding cost negates this advantage. On the other side, a higher value of $k$ leaves the tree far from the optimum, causing search to be rather expensive and impacting on the overall performance.

## 5 Conclusion

Stochastic simulation is an emerging research area for investigating bio-inspired processes, especially whenever fluctuation and noise play an important role. Gillespie's SSA has become a *de facto* standard for simulating the biochemical reaction systems. In general,

the stochastic simulation is composed of two steps: finding the next reaction firing, and updating the system state. In this paper we apply binary search in trees to the SSA. In particular, we exploit the Huffman tree to reduce the number of comparisons needed for finding the next reaction firing. We proposed two strategies for keeping the tree close to the Huffman optimum during simulation, and studied their performance. In our example the adaptive tree rebuilding appears to be more promising than the periodic updating.

Several improvements for the future are possible. For instance, in the studied approaches we either leave the Huffman tree as it is, or perform a complete rebuilding. One could then imagine to interleave full rebuilding, which is expensive, with a cheaper partial optimization. The latter would not restore the tree to an optimal case, but just improve it slightly. For instance, if a deep node in the tree is found to have higher propensity than a shallow node, we can quickly swap them and improve the tree. This optimization mechanism would be then similar to those used in garbage collection in computing systems, which is often split in frequent minor collections and rare major ones. As another approach, one could even explore the use of $n$-ary trees instead of binary trees.

Various methods to efficiently search for the next reaction firing are found in the literature, such as those described in Hormann et al. (2004). For example, a lookup table (or guide table), instead of sequential or binary search, could be used to drive the search in constant time; however, its main drawback is the requirement to set up a large support table after each reaction firing, which appears to negate the advantage. Overall, we believe there is no optimal solution for all models, and combining different methods can be a promising research line for the future.

## References

Alonso, S. and Sagués, F. and Mikhailov, A. S. (2006) 'Negative-tension instability of scroll waves and winfree turbulence in the oregonator model', *J. Phys. Chem. A*, Vol. 110, No. 43, pp.12063–12071

Blue, J. and Beichl, I. and Sullivan, F. (1995) 'Faster Monte Carlo simulations', *Phys. Rev. E*, Vol. 51, No. 2, pp.867–868

Cao, Y. and Li, H. and Petzold, L. (2004) 'Efficient formulation of the stochastic simulation algorithm for chemically reacting systems', *J. Chem. Phys.*, Vol. 121, No. 9, pp.4059–67

Knuth, D. (1968) *The Art of Computer Programming*, Vol. 1, Addison-Wesley.

Cao, Y. and Gillespie, D. T. and Petzold, L. (2006) 'Efficient step size selection for the tau-leaping simulation method', *J. Chem. Phys.*, Vol. 124, No. 4, pp.44109

Crampina, E. and Schnella, S. and McSharry, P. (2004) 'Mathematical and computational techniques to deduce complex biochemical reaction mechanisms', *Progress in Biophysics and Molecular Biology*, Vol. 86, No. 1, pp.77—112

Dittamo, C. and Cangelosi, D. (2009) 'Optimized parallel implementation of Gillespie's first reaction method on graphics processing units', *in Proc. of ICCMS*, pp.156–161

Gibson, M. A. and Bruck, J. (2000) 'Efficient exact stochastic simulation of chemical systems with many species and many channels', *J. Phys. Chem. A*, Vol. 104, No. 9, pp.1876–1889

Gillespie, D. T. (1976) 'A general method for numerically simulating the stochastic time evolution of coupled chemical reactions', *J. Comp. Phys.*, Vol. 22, No. 4, pp.403–434

Gillespie, D. T. (1977) 'Exact stochastic simulation of coupled chemical reactions', *J. Phys. Chem.*, Vol. 81, No. 25, pp.2340–2361

Gillespie, D. T. (1992) 'A rigorous derivation of the chemical master equation', *Physica A*, Vol. 188, pp.404–425

Gillespie, D. T. (2001) 'Approximate accelerated stochastic simulation of chemically reacting systems', *J. Chem. Phys.*, Vol. 115, pp.1716–1733

Gillespie, D. T. (2007) 'Stochastic simulation of chemical kinetics', *Annu. Rev. Phys. Chem.*, Vol. 58, pp.35–55

Gillespie, D. T. and Petzold, L. (2003) 'Improved leap-size selection for accelerated stochastic simulation', *J. Chem. Phys.*, Vol. 119, No. 16, pp.1716–1723

McAdams, H. and Arkin, A. (1997) 'Stochastic mechanisms in gene expression', *in Proc. of PNAS*, Vol. 94, No. 9, pp.814–819

McAdams, H. and Arkin, A. (1999) 'It's a noisy business! Genetic regulation at the nanomolar scale', *Trends in Genetics*, Vol. 15, No. 2, pp.65–69

Hormann, W. and Leydold, J. and Derflinger, G. (2004) *Automatic Nonuniform Random Variate Generation*, Springer-Verlag.

Huffman, D. A. (1952) 'A method for the construction of minimum-redundancy codes', *in Proc. of IRE*, pp.1098–1101

Indurkhya, S. and Beal, J. (2010) 'Reaction factoring and bipartite update graphs accelerate the Gillespie Algorithm for large-scale biochemical systems', *PLoS One*, Vol. 5, No. 1, pp.8125

Kolch, W. (2000) 'Meaningful relationships: the regulation of the ras/raf/mek/erk pathway by protein interactions', *Biochem. J.*, Vol. 351, No. 2, pp.289–305

Kurata, H. and El-Samad, H. and Yi, T. M. and Khammash, M. and Doyle, J. (2001) 'Feedback regulation of the heat shock response in E. coli', *in Proc. of CDC*, Vol. 1

Li, H. and Petzold, L. (2006) *Logarithmic direct method for discrete stochastic simulation of chemically reacting systems.*, Technical Report. http://engineering.ucsb.edu/ cse/Files/ldm0513.pdf (Accessed 22 Feb. 2013).

Li, H. and Petzold, L. (2010) 'Efficient Parallelization of the Stochastic Simulation Algorithm for Chemically Reacting Systems On the Graphics Processing Unit', *International Journal of High Performance Computing Applications*, Vol. 24, No. 2, pp.107-116

Mauch, S. and Stalzer, M. (2011) 'Efficient formulations for exact stochastic simulation of chemical systems', *IEEE/ACM Trans. on Computational Biology and Bioinformatics*, Vol. 8, No. 1, pp.27–35

McCollum, J. M. and Petersonb, G. D. and Cox, C. D. and Simpson, M. L. and Samatova, N. F. (2006) 'The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior', *Comput. Bio. Chem.*, Vol. 30, No. 1, pp.39–49

Pedraza, J. and van Oudenaarden, A. (2005) 'Noise propagation in gene networks', *Science*, Vol. 307, No. 5717, pp.1965–1969

Pedraza, J. and van Oudenaarden, A. (2005) 'Exact and Approximate Stochastic Simulations of the MAPK Pathway and Comparisons of Simulations Results', *Journal of Integrative Bioinformatics*, Vol. 3, No. 2, pp.38

Raser, J. M. and O'Shea, E. K. (2005) 'Noise in gene expression: Origins, consequences, and control', *Science*, Vol. 309, No. 5743, pp.2010–2013

Ramaswamy, R. and González-Segredo, N. and Sbalzarini, I. F. (2009) 'A new class of highly efficient exact stochastic simulation algorithms for chemical reaction networks', *J. Chem. Phys.*, Vol. 130, No. 24, pp.244104

Schulze, T. P. (2008) 'Efficient kinetic monte carlo simulation', *J. Comp. Phys.*, Vol. 227, No. 4, pp.2455–2462

Slepoy, A and Thompson, A. P. and Plimpton, S. J. (2008) 'A constant-time kinetic Monte Carlo algorithm for simulation of large biochemical reaction networks', *J. Chem. Phys.*, Vol. 128, No. 20, pp.205101

Thanh, V. H. and Zunino, R. (2011) 'Parallel stochastic simulation of biochemical reaction systems on multi-core processors', *in Proc. of CSSim*

Thanh, V. H. and Zunino, R. (2012) 'Tree-Based Search for Stochastic Simulation Algorithm', *in Proc. of ACM SAC*

Vilar, J. and Kueh, H. and Barkai, N. and Leibler, S. (2002) 'Mechanisms of noise-resistance in genetic oscillators', *in Proc. of PNAS*, Vol. 99