Department of
Information Engineering
and Computer Science
DISI

UNIVERSITY
OF TRENTO - Italy

DISI - Via Sommarive 14 - 38123 Povo - Trento (Italy)
http://www.disi.unitn.it

TREE-BASED SEARCH FOR STOCHASTIC SIMULATION ALGORITHM

Vo Hong Thanh, Roberto Zunino

November 2011

Technical Report # DISI-11-478

Tree-Based Search for Stochastic Simulation Algorithm

Vo Hong Thanh
University of Trento, Italy
vo@disi.unitn.it

Roberto Zunino
University of Trento, Italy and COSBI
zunino@disi.unitn.it

Abstract

In systems biology, the cell behavior is governed by a series of biochemical reactions. The stochastic simulation algorithm (SSA), which was introduced by Gillespie, is a standard method to properly realize the dynamic and stochastic nature of such systems. In general, SSA follows a two-step approach: finding the next reaction firing, and updating the system accordingly. In this paper we apply the Huffman tree, an optimal tree for data compression, so to improve the search for the next reaction firing.

Keywords: Systems biology, SSA, Tree search SSA, Huffman tree SSA.

1 Introduction

In recent years, the modeling and simulation of biological systems has become an emergent research field in computational systems biology. Earlier approaches use deterministic models such as ordinary differential equations (ODE) to analyze the behavior of systems. Recent research has shown the importance of fluctuation and noise in e.g. gene expression and biochemical networks [18]. Hence, instead of considering deterministic changes of the concentration of each reactant, the stochastic approach tries to realize the dynamics in the population of each species in the system.

The stochastic simulation algorithm (SSA) [7, 8] is an exact simulation method for simulating and reproducing the intrinsic noise of chemical reactions. The evolution of the system is computed by firing one reaction at each step. SSA uses a Monte Carlo simulation technique to sample the system state, where the firings of each reaction define a Poisson process with propensity (or rate) a_j. There are two well-known implementations of the SSA method, namely the First Reaction Method (FRM) and the Direct Method (DM). DM and FRM are mathematically equivalent procedures 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. By contrast, DM generates the next firing time, and then a search is conducted to find the fired reaction. The system is updated accordingly after that, and the algorithm proceeds with the next simulation step.

Many improvements have been introduced to both FRM and DM. The Next Reaction Method (NRM) [5] improves FRM by representing the dependencies among reactions using a graph, and employing a special indexed structure to store putative times. Hence, less time is spent for choosing a reaction and updating the system state. The Optimized Direct Method (ODM) [22] and Sorting Direct Method (SDM) [17] improve DM based on the observation that the search will be faster when reactions are indexed according to their frequencies. The Logarithmic Direct Method (LDM) [15] speeds up the simulation by applying binary search instead of sequential search in DM.

Variants of SSA by dividing reactions into groups have been proposed in [16, 19]. In [16] it is proposed a 2D search for finding the next reaction, storing the propensities in a matrix. The next firing reaction is found by two linear searches: one finds the row using partial sums, and the other finds the reaction within the row. The method in [19] combines the sequential search in a first stage and rejection method in a second stage for choosing the reaction.

Recently, the computational power of parallel and distributed environment has been exploited to improve the performance of SSA. There are two main types of parallel techniques for simulating the dynamic behavior of the cell: simulation parallelism and functional parallelism. The former will execute the same simulation multiple times on each logical process to generate many samples of the system [15]. In the latter instead the simulation algorithm is divided into functional components and a number of logical processes will simulate for corresponding components [4, 20].

Beyond exact methods, various approximated methods have been presented to reduce the simulation time, such as [9]. For example, the τ-leaping method (more details in [2,6,10]) 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. Approximated methods greatly speed up the simulation. However, the accuracy of the results sometimes is uncertain, and care must be taken to avoid abnormal results (e.g., negative populations).

Improving the exact method, therefore, is still useful. In this work, we describe in detail a tree-based data structure and its implementation for improving the original SSA method. Tree-based structures allow binary search, which is of course more efficient than the linear search used in many variants of SSA. However, using a complete binary tree may not lead to an optimal performance, i.e. it might not lead to a minimum number of comparisons performed during search. Indeed, in this work we study how to use the Huffman tree for this purpose, and discuss some experimental findings.

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 discuss, and some possible future research directions are provided in section 5.

2 SSA

We consider a well-stirred chemical 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:

$$R_j : v_{1j}S_1 + ... + v_{nj}S_n \stackrel{k_j}{\rightarrow} v'_{1j}S_1 + ... + v'_{nj}S_n \tag{1}$$

where v_{ij} and v'_{1j} are the *stoichiometric coefficients* and k_j is the *rate constant* of reaction R_j. In the case the system contains reversible reactions, they can be modelled as two irreversible reactions.

The dynamical 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 ith element equals to $v'_{ij} - v_{ij}$, which describes the change in the population of species S_i. Therefore, the state transition of the system 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 $P(x, t)$ given the initial state $X(t_0) = x_0$ at time t_0.

$$\begin{aligned} \frac{\delta P(x, t|x_0, t_0)}{\delta t} &= \sum_{j=1}^{m} a_j(x - v_j)P(x - v_j, t|x_0, t_0) \\ &\quad - \sum_{j=1}^{m} a_j(x)P(x, t|x_0, t_0) \end{aligned} \tag{3}$$

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

Equation 3 is generally called chemical master equation (CME). The CME completely determines the time evolution of the system. An analytic solution of this equation is hard to find, unless the system is very small. However, although the CME equation cannot be solved in general, the evolution of the system state in time can be computed through simulating the joint probability density function $p(\tau, j|x, t)$, which is the probability the reaction R_j will be fired in the time interval $[t + \tau, t + \tau + dt)$, provided we are in state $X(t) = x$.

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

where

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

where τ and j are the time of the reaction firing and its index, respectively.

Based on 4-5, the Direct Method (DM) computes τ and j by generating two random numbers r_1 and r_2 from the uniform distribution $U(0, 1)$, and then applies the inversion

method to obtain the values of τ and j:

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

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

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

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

The smallest value τ_j is used, and the corresponding reaction R_j is chosen. The general schema of the SSA algorithms for the exact simulation of biochemical reaction systems is given in Algorithm 1.

Algorithm 1 SSA

1: initialize system time $t = t_0$ and system state $x = x_0$
2: **for all** reaction channel R_j **do**
3: compute a_j and their sum a_0
4: **end for**
5: generate the values of τ and j according to chosen method (FRM, DM)
6: update the time $t = t + \tau$ and system state $x = x + v_j$
7: goto step 2

2.1 Next Reaction Method (NRM)

This method is an improvement of FRM. It uses a priority queue, which is typically implemented as a binary heap, to store the putative times of reactions so that retrieving the smallest one can be performed efficiently. Moreover, care is taken for handling dependent reactions: firing one reaction can change the amount of reactants for other reactions, and so affect the propensity which has to be recomputed. Further, in that case, the putative times which are stored in the priority queue have to be adjusted accordingly. NRM pre-computes a reaction dependency graph to perform these steps efficiently, such as that in Fig. 1.

We denote with affects(R_j) the set of reactions that R_j affects. Formally:

$$\begin{aligned} \text{affects}(R_j) = \{ \, & R_i \mid \\ & (\text{reactants}(R_j) \cup \text{products}(R_j)) \cap \text{ reactants}(R_i) \neq \emptyset \} \end{aligned} \tag{9}$$

where reactants(R_j) and products(R_j) are the set of species taking part in reaction R_j as reactants and products, respectively. The directed dependency graph $DG(V, E)$ contains as vertices V all the reactions, while we find an edge $e(R_i, R_j) \in E$ if and only if $R_j \in \text{affects}(R_i)$. When one reaction fires, NRM searches for neighbors on the dependency graph, so to update their putative times.

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

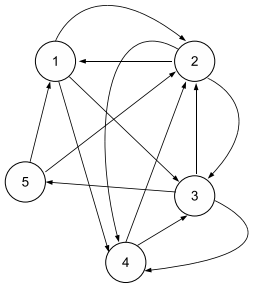

Figure 1: Dependency graph (removing self affected edges)

3 Binary Search for SSA

Binary search is a more efficient search method than the linear search which is used in DM (logarithmic vs. linear complexity). It was proposed in LDM to improve DM exploiting the partial sums of propensities. However, the underlying data structure was not completely explained. In addition, LDM did not use a dependency graph for updating the system, which could further improve the performance. In this section, we first detail the data structures and algorithms used to apply binary search on a complete tree to SSA. Then, we switch from complete trees to Huffman trees, so to exploit their optimal 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 as exactly two children (internal node), or zero (leaf). For our purposes, leaves hold the propensities of reactions, while internal nodes store the sums of values of their child nodes. Theorem 1 and the following discussion allow to store a complete tree on a contiguous array, hence improving cache-friendliness.

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

Proof Denote with s the number of internal nodes. In a complete tree each internal node has two child nodes. Hence, the number of edges in the tree is $2s$. Also, the edges are $m + s - 1 = 2s$, hence $s = m - 1$. So, the number of nodes is $s + m = 2m - 1$.

Therefore, we can use an array with $2m - 1$ elements to represent a complete tree with m reaction propensities 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 can recursively construct the tree with the internal sums as follows.

In Algorithm 2, each element of the array TREE stores the partial sums of the reaction propensities, so we simply need each array cell to store a simple type (a floating point double). In order to build the tree, the number of reactions m must be an even

Algorithm 2 Building the complete tree

procedure: *build_tree*(position)

require: array TREE with $2m-1$ elements where elements from $m-1$ 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
```

number. In the case m is not even, we can add a dummy reaction (with propensity 0) as the last element of the array.

After 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 (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 from it the value stored in the left branch. The whole procedure is outlined in Algorithm 3. The procedure is correct, in the sense that it finds the same leaf j as in (7), so each reaction indeed can be fired with the correct probability a_j/a_0.

Algorithm 3 Finding the next reaction firing

procedure: *search*(position, v)

require: properly set up array TREE, search value v

```
1: while position is not leaf do
2:     if TREE[2position] ≥ v then
3:         search(2position, v)
4:     else
5:         v = TREE[position] - v
6:         search(2position + 1, v)
7:     end if
8: end while
```

When updating the system state, we need to update the propensity tree as well. For that, we use a dependency graph and exploit the fact that the parent of node i is located at position $\lfloor i/2 \rfloor$. Therefore, we only need to update the affected reactions and their ancestor nodes in the tree.

3.2 Huffman Tree Search

While storing the reactions in a complete tree minimizes the *height* of the tree, 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. Denoting this number with $T_m(C)$, we have:

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

where m is the total number of reactions, w_j is the weight of reaction j, and D_j is the depth of the leaf in the tree corresponding to reaction R_j, respectively. The weight w_j is the probability of reaction R_j being selected to fire.

In the complete tree approach, 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 comparisons 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 $T_m(C) = 2$. Using 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 $T_m(C) = 1.18$ comparisons, which is better. Intuitively, we can improve the performance of the complete tree search, especially for multi-scale 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.

The above 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. In [12], Huffman described a possible way to construct the tree so to minimize $T_m(C)$. The basic 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 $w_{pq} = w_p + wq$. 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:

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

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

Theorem 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.

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

Algorithm 4 Building Huffman tree

procedure: *build_huffman_tree*

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

1: build heap H with elements $(m-1, w_1)$,... $(2m-1, w_m)$, ordered according to w_j
2: **for** $position = m-2$ 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**

The Huffman tree we build in the Algorithm 4 is stored in an array in which each cell contains the fields VALUE, LEFT, and 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 field, respectively. The same binary search procedure of Algorithm 3 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.

3.3 The weight function and tree updates

Using the Huffman tree, we want to reduce the number of comparisons to find the next firing reaction. A good candidate for the weight function is the propensity function a_j since this choice leads to less time spent for finding the fast reaction (which have large propensity).

However, during the execution of the SSA, reaction firings affect the propensity of reactions, which can also change rapidly. This happens, for example, whenever the reaction has a very large rate constant but a small number of reactant molecules. When it is fired, its propensity significantly changes by a large amount. When propensities change, we need to update the values stored in the tree, as we did using the complete trees. This update, however, 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. For this reason we choose to keep using a non-optimal tree for some predefined (and tunable) number of SSA steps, postponing the reconstruction of the tree after those steps. Note that the choice of this parameter only affects the performance, while the results are still exact.

Further, to cope with propensities changing rapidly, we slightly modify the weights w_j so to assign a higher weight to those reactions which are more likely to change. For each reaction R_j, we define sets conflicts(R_j) as the collection of reactions that affect or compete with R_j

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

and favors(R_j) is the collection of reactions that affects and favors R_j

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

In terms of the dependency graph $DG(V, E)$, we have the following relationship: |conflicts(R_j)| + | favors(R_j)| = in-degree(R_j).

Under these definitions, when a reaction fires, we can estimate the probability it will increase (decrease) the propensity of reaction R_j as |conflicts(R_j)|$/m$ (|favors(R_j)|$/m$). After 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 α_1, α_2 are parameters denoting the average change amount. For simplicity, we assign it to the stochastic rate constant for the reaction at hand i.e, $\alpha_1 = -\alpha_2 = k_j$.

To update the propensity of affected reactions, we adapt the Huffman tree building procedure by storing location of parent node in an additional field, called PARENT, in each element of array TREE. With this information we can build the path from a leaf to the root and reflect the changes.

4 Experimental Results

In this section, we report the simulation results for various models differing in size. The table 1 provides a summary of the number of reactions and species of the simulated systems.

Table 1: List of tested systems 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

4.1 Tested Models

The smallest model is the Oregonator 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 [1]. The species and reactions of Oregonator model are shown in table 1.

For the second test, we use the simplified circadian cycle model in [21]. The circadian rhythm is a daily cycle in the biochemical processes of many living beings and is widely observed in plants and animals, so providing a sort of internal clock and regulate behavior accordingly. The key mechanism of the circadian rhythm is the intracellular transcription regulation of two genes, an activator and a repressor. Both are translated into activator and repressor proteins. Activator acts as the positive element in transcription in binding to promoter, while repressor acts as the negative element by repressing the activator.

In the third model, we simulate the heat shock response (HSR) process occurring 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 is controlled by the so-called sigma factor. The sigma factor 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 $\sigma 32$ will replace $\sigma 70$, which is the bound σ unit of RNA Polymerase (RNAP), to accelerate HSPs synthesis (more detais in [14]).

The largest 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 responds a response pattern including 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 of this pathway is constructed by three protein kinases: MAPKKK (such as RAS/Raf), MAPKK (such as MEK) and MAPK. The external stimuli activate the first element the MAPKKK of the pathway. 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 (more details in [13]).

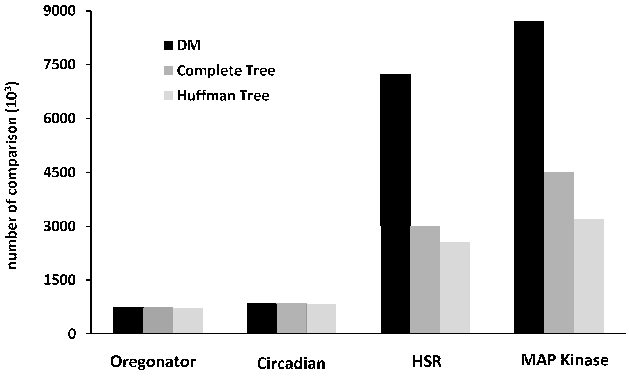

Figure 2: Comparisons performed by each algorithm

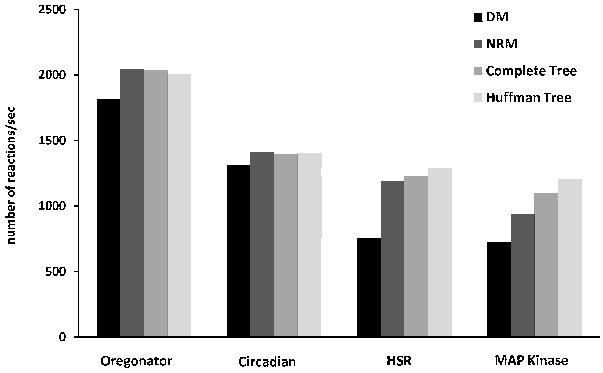

Figure 3: Overall performance

4.2 Performance

The performance of our algorithm is reported in Fig. 2-3. 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 for updating the propensities of the affected reactions. For the Huffman Tree Search, we had to pick a number k of steps after which we reconstruct the Huffman Tree, as we discussed in Sect. 3.3. Picking a too small or too large value for k may degrade the overall performance. In our simulation, we chose $k = 100,000$, so causing the tree to be rebuilt 5 times in the whole simulation.

In Fig. 2 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 queue for selecting. In all the samples, the Huffman tree search performed the least number of comparisons. When simulating small models, the difference between linear search and binary search is not very significant.

However, when using the bigger models binary search is nearly 50% faster than linear search, and Huffman Tree Search still gain another $\sim 20\%$ in comparisons with respect to Complete Tree Search.

As shown in Fig. 3, simulating 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.

5 Conclusions

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 cell behavior. In general, the stochastic simulation of biochemical reaction systems is composed of two steps: finding the next reaction firing, and updating the system state. In this paper we apply the Huffman tree to the SSA to reduce the number of comparisons to find the next reaction firing. By estimating the change of the weights in the Huffman tree, we can minimize the overhead due to the need of rebuilding the tree.

In fact, in the literature we can find various methods to efficiently choose the next reaction firing, such as those described in [3, 11]. For example, instead of sequential or binary search a lookup table (or guide table) could be used to drive the search. However, its main drawback is the requirement to set up a large table after each reaction firing. Overall, there is no optimal solution for all models, and a combination of methods into the current stochastic simulation algorithm can be a promising approach for the future.

References

[1] S. Alonso, F. Sagus, and A. S. Mikhailov. Negative-tension instability of scroll waves and winfree turbulence in the oregonator model. *J. Phys. Chem. A*, 110(43):12063–12071, 2006.

[2] Yang Cao, Daniel T. Gillespie, and Linda R. Petzold. Efficient step size selection for the tau-leaping simulation method. *J. Chem. Phys.*, 124(4):044109, 2006.

[3] Luc Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, 1986.

[4] C. Dittamo and D. Cangelosi. Optimized parallel implementation of Gillespie's first reaction method on graphics processing units. In *Proc. of ICCMS*, pages 156-161, 2009.

[5] Michael A. Gibson and Jehoshua Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *J. Phys. Chem. A*, 104(9):1876-1889, 2000.

[6] Daniel T. Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *J. Chem. Phys.*, 115:1716-1733, 2001.

[7] Daniel T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *J. Comp. Phys.*, 22(4):403-434, 1976.

[8] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81(25):2340-2361, 1977.

[9] Daniel T. Gillespie. Stochastic simulation of chemical kinetics. *Annu. Rev. Phys. Chem.*, 58:35-55, 2007.

[10] Daniel T. Gillespie and Linda R. Petzold. Improved leap-size selection for accelerated stochastic simulation. *J. Chem. Phys.*, 119(16):1716-1723, 2003.

[11] Wolfgang Hormann, Josef Leydold, and Gerhard Derflinger. *Automatic Nonuniform Random Variate Generation*. Springer-Verlag, 2004.

[12] David A. Huffman. A method for the construction of minimum-redundancy codes. In *Proc. of IRE*, volume 40, pages 1098-1101, 1952.

[13] W Kolch. Meaningful relationships: the regulation of the ras/raf/mek/erk pathway by protein interactions. *Biochem. J.*, 351(2):289-305, 2000.

[14] H. Kurata, H. El-Samad, T.-M. Yi, M. Khammash, and J. Doyle. Feedback regulation of the heat shock response in E. coli. In *Proc. of CDC*, volume 1, 2001.

[15] Hong Li and Linda Petzold. Logarithmic direct method for discrete stochastic simulation of chemically reacting systems. Technical Report, 2006.

[16] Sean Mauch and Mark Stalzer. Efficient formulations for exact stochastic simulation of chemical systems. *IEEE/ACM Trans. on Computational Biology and Bioinformatics*, 8(1):27-35, 2011.

[17] James M. McCollum, Gregory D. Peterson, Chris D. Cox, Michael L. Simpson, and Nagiza F. Samatova. The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior. *Comput. Bio. Chem.*, 30(1):39-49, 2006.

[18] Jonathan M. Raser and Erin K. O'Shea. Noise in gene expression: Origins, consequences, and control. *Science*, 309(5743):2010-2013, 2005.

[19] Tim P. Schulze. Efficient kinetic monte carlo simulation. *J. Comp. Phys.*, 227(4):2455.

[20] Vo H. Thanh and Roberto Zunino. Parallel stochastic simulation of biochemical reaction systems on multi-core processors. In *Proc. of CSSim*, 2011.

[21] J. Vilar, H. Kueh, N. Barkai, and S. Leibler. Mechanisms of noise-resistance in genetic oscillators. In *Proc. of PNAS*, volume 99, 2002.

[22] Linda R. Petzold Yang Cao, Hong Li. Efficient formulation of the stochastic simulation algorithm for chemically reacting systems. *J. Chem. Phys.*, 121(9):405967, 2004.