Polymers in Kappa: a tutorial

Jean Krivine [*]

PPS lab. CNRS and Univ. Paris Diderot - UMR 7126

1 Introduction

In this tutorial we want to study the question of how a protein with a specific DNA interaction capability, such as a transcription factor, is able to (rapidly) find its target on DNA. Indeed if one considers that a DNA strand has the order of 10^9 base pairs, finding a particular sequence of a few nucleotides by colliding with DNA at random is rather unlikely. It has been argued [2] that enzymes with specific DNA interactions combine 3D search (diffusion on the cytoplasm with random encounter with DNA) with 2D or 1D diffusion along the DNA to find their substrate (see Fig. **??**).

In the present tutorial, we show that Kappa provides mechanistic ways to model such systems. We will model both 2D and 3D diffusion of a repair enzyme that is looking for U/G mismatches on DNA that arise after spontaneous hydrolytic deamination of cytosine bases. Deamination corresponds to the chemical transformation of a cytosine into a uracil. It is estimated that in a given genome, between 50 to 500 cytosines are deaminated per day [3]. Several enzymes are known to have the capacity to recognize and initiate the repair of U/G mismatches, namely UNG, SMUG1, TDG and MBD4. In the present tutorial we are not interested in the specific modeling of these enzymes and we will simply consider UDG (Uracil DNA Glycosylase) from the UNG family, the most frequent one. Roughly the repair process is the following: UDG, probably after a combination of sliding and 3D diffusion, reaches a U/G mismatch on DNA. It anchors strongly to it and excises the faulty U base, leaving an apurinic (AP) site: the DNA backbone with no nucleotide. The remaining part of the repair process is called *Base Excision Repair* and involves several other enzymes that are able to put a cytosine back in the AP site, re-glue the DNA backbone and reform the C:G pairing.

[*]Jean.Krivine@pps.univ-paris-diderot.fr

The model we will consider here stops after the intervention of UDG and we will treat the remaining of the repair as a (unbiological) single step.

There are three modeling challenges to raise here. The first one is of combinatorial nature: if one wishes to represent DNA sliding on a strand of size n_D, then one must take into account the fact all possible positions of the repair enzyme on DNA constitute a different molecular species, ie. there is the order of 2^{n_D} possible configurations. Also, if all the n_C cytosine bases present on a DNA may undergo a repair cycle of the form $C \to U \to AP \to C$, one must also consider the additional 3^{n_C} additional configurations of DNA. Essentially any approach which would try to enumerate (even on the fly) possible configuration would be doomed to fail. This first challenge is particularly well suited for Kappa, whose simulator has a cost per event that is independent of the number of possible molecular species the system may generate [1]. However using Kappa raises a second problem, inherent to the fact that space is abstracted away in this formalism. We will deal with this issue by showing one may use the structure link structure of DNA to model faithfully the distance between the enzyme on DNA and its substrates. Last, one has to face the question of running simulations on a realistic sample of DNA. We will see in the preliminary part of this tutorial, that one may use KaSim to generate meaningful initial conditions.

2 Preliminaries: generating initial conditions

In this part of the tutorial one is interested in generating a Kappa representation of a double DNA strand with a random (and uniform) distribution of base pairs.

2.1 The choice of agent signatures

The obvious way to go would be to use one agent per nucleotide A,T,G,C or U. However a better choice is to consider a generic agent N (for nulceotide) which will allow us to specify rules which apply to all types of nucleotides (see Figure ??). The signatures of N and UDG are declared in KaSim by the lines:

```
%agent : N(e3, u, e5, o, i~A~T~G~C~U~AP)
%agent : UDG(dbd)
```

where `e3` and `e5` are respectively the 3' and 5' ends of the nucleotides, `o` is its "outter" face and `i` its "inner" face. Note that this latter has 5 possible internal states which will be used to specialize the nucleotide to an A,T,G or C base, or an apurinic site. We equip UDG with a *DNA binding domain* site called `dbd`.

2.2 Growing DNA

In a file `synth.ka` we put the signatures defined in the previous section, together with the following rules that will enable us to grow DNA strands:

```
'growAT'  N(i!0, e5), N(i!0, e3) -> \
          N(i!0, e5!1), N(i!0, e3!2), N(e3!1, i~A!3), N(e5!2, i~T!3)@'kAT'
'growTA'  N(i!0, e5), N(i!0, e3) -> \
          N(i!0, e5!1), N(i!0, e3!2), N(e3!1, i~T!3), N(e5!2, i~A!3)@'kAT'
'growCG'  N(i!0, e5), N(i!0, e3) -> \
          N(i!0, e5!1), N(i!0, e3!2), N(e3!1, i~C!3), N(e5!2, i~G!3)@'kGC'
'growGC'  N(i!0, e5), N(i!0, e3) -> \
          N(i!0, e5!1), N(i!0, e3!2), N(e3!1, i~G!3), N(e5!2, i~C!3)@'kGC'
```

Note that these rule may grow a new base pair provided there pre-exists in the mixture a base pair with a free 3':5' end. One solution would be to start from an initial state with such a pair. However a more elegant solution is seed the synthesis of DNA with the rules:

```
'seedAT'  -> N(i~A!0), N(i~T!0) @ 'sAT'
'seedGC'  -> N(i~G!0), N(i~C!0) @ 'sGC'
```

Recall that the convention adopted in KaSim is that agents that are created by a rule and whose interface is not fully specified, are completed using the signature of the agent[1].

In DNA, the ratio of GC pairs over AT ones is not evenly distributed. In the human genome the GC content (defined as $\frac{G+C}{A+T+G+C}$) is about 40%.

Question 1: Assuming that `'sAT'`=`'kAT'`, what should be the value of `'kGC'` and `'sGC'` in order to respect this ratio?

Answer: The probability that rule r applies is $\alpha(r)/\sum_i \alpha(r_i)$, where $\alpha(r_i)$ is the activity of rule r_i defined as the number of matches of its left hand side, times its kinetic rate. Note that all the rules that introduce pairs of nucleotide have equal left hand sides. Therefore we can easily show that in order to respect the 40% ratio of GC content, it suffices to set $\alpha($`'growAT'`$)/\alpha($`'growGC'`$) = \alpha($`'seedAT'`$)/\alpha($`'seedGC'`$) = 1.5$.

[1] whenever a site that has possible internal states are not specified in the created agent, it receives the value indicated first in the signature.

In KaSim variables can be declared by:

```
%var: 'kGC' 1.0
%var: 'kAT' 1.5 * 'kGC'
%var: 'sGC' 1.0
%var: 'sAT' 1.5 * 'sGC'
```

Question 2: Given a state with n strands of growing DNA. What is the likelihood that that a given strand is the next one to grow?
Answer: Any strand has exactly two matches for the growing rules, one on the 3':5' end and the other on the 5':3' end. Therefore the probability that a given strand is the next to grow does not depend on its current length and is uniformly distributed. As a result one should expect that the n strands will have in average the same length.

Question 3: One wishes to favor growing strands over creating new ones. What can we do for this?
Answer: an easy solution is to increase the ratio `'kXY'`/`'sXY'`. For instance we can set `'kGC'` = 10 and `'sGC'` = 1.

2.3 Controlling the number of strands

Instead of growing an undetermined number of DNA strands one may wish to restrict to *exactly* n strands. To do so, one needs to count how many times the seed rules have been applied. We will use *tokens* to do this. We add the signature:

```
%token: Cpt
```

and patch the seed rule as:

```
'seedAT' -> N(i~A!0), N(i~T!0) | 1 : Cpt @ 'sAT'
'seedGC' -> N(i~G!0), N(i~C!0) | 1 : Cpt @ 'sGC'
```

which tells KaSim to create 1 token `Cpt` at each application of a seed rule. Now the total number of seed applications can be accessed by `|Cpt|` (the number of tokens `Cpt`). One may then use the perturbation:

```
%mod: |Cpt| = n do ($UPDATE 'sAT' 0. ; $UPDATE 'sGC' 0.)
```

which will turn off seeding new strands after n applications. Note that since `'sAT'`=0 if `'sGC'`=0 it suffices to use the perturbation:

```
%mod: |Cpt| = n do $UPDATE 'sGC' 0.
```

2.4 Controlling the simulation

In order to check that the GC content of the mixture is respected, one defines the following variables:

```
%var: 'C' N(i~C?)
%var: 'G' N(i~G?)
%var: 'N' N()
```

which enable us to define the observable:

```
%obs: 'GC content' ('C' + 'G')/'N'
```

Now one may run KaSim on the `synth.ka` file using the command:

```
KaSim -i synth.ka -e 100000 -p 1000 -o data.out
```

which will generate one strand (if $n = 1$ in the perturbation) of DNA of 10^5 base pairs.

The content of `data.out` can be plotted using gnuplot for instance, and the result is given Fig. 1.

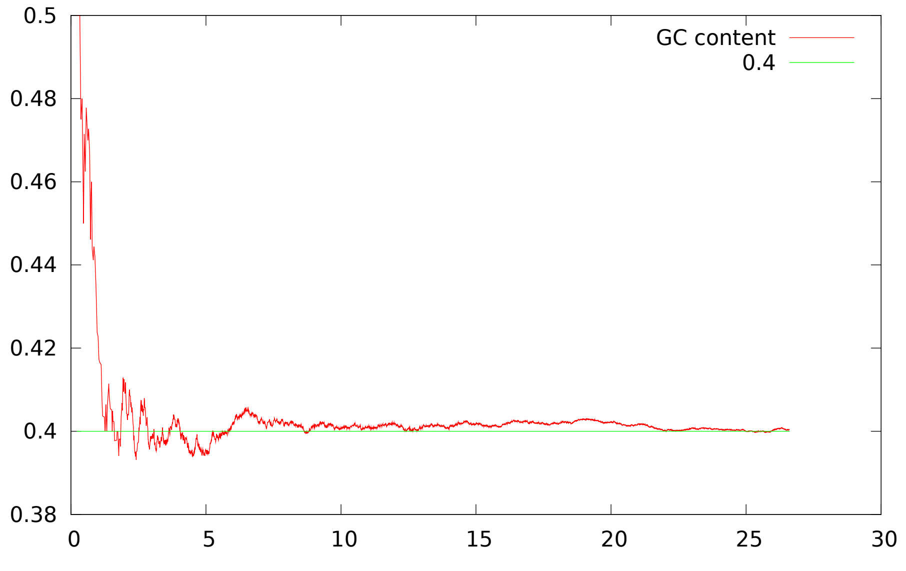

Figure 1: Controlling the GC content of the synthesized DNA.

2.5 Visualizing and saving the final state

Now that the model seems to be generating DNA, we need to be able to extract the final state of the simulation and use it as one of the input of our DNA repair model.

In order to save the state of a simulation one uses the perturbation:

```
%mod: 'N' = n do $SNAPSHOT ''dna.ka''
```

which asks KaSim to save the state[2] whenever the number of nucleotides has reached n (which makes a double strand DNA of length $n/2$). The state will be saved in the file `dna.ka` (which does not contain the signature of the agents).

It is often convienent to generate a snapshot of the state that one may visualize instead of a plain Kappa file. To do so, one may tell KaSim to generate a dot format snapshot using the command:

```
%def: ''dotSnapshots'' ''true''
```

and the snapshot will be saved in the dot format file `dna.dot` instead. Dot format may be vizualised using the free software graphviz (and others). The snapshot of our model, using $n = 20$ is given Fig.2.

3 The DNA repair model

We leave aside for now the `dna.ka` file generated at the previous state. We wish to represent in a different file, say `repair.ka`, the different step of the DNA repair model.

3.1 Deamination

Question 1: Assuming there are about 500 deamination events occuring each day in a mamalian cell (mamalian genomes have the order of 10^9 base pairs). What should be the deamination rate of a single cytosine per second?
Answer: We saw that the GC content of a mamalian cell is 40%. Hence there are about 20% cytosines in a given genome, which makes $n_C = 2 * 10^8$ cytosines. We have $n_C * k = 500$ where k is the deamination rate per cytosine

[2]The snapshot mechanism of KaSim will regroup all isomorphic species together and output each isomorphic class in an `%init:` k E declaration, where k is the number of elements in the class, and E the Kappa expression of one representative of the class.

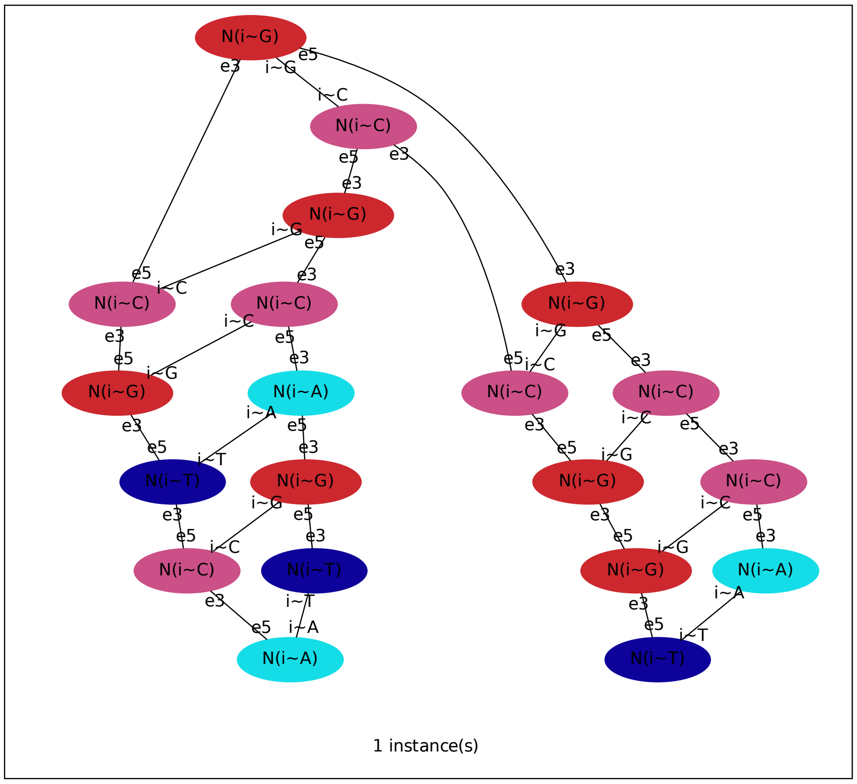

Figure 2: Outputting the snapshot as a dot format enables one to visualize the species that are formed during a simulation, here an ACTGCGGCCA strand.

per day. So the rate per second is: $k_{deam} = 500/(n_C * 24 * 60 * 60) = 8.64 * 10^{-11} s^{-1}$.

Obviously, this rate being very low, we will have to increase it a lot if we want to observe deamination in our small DNA strand.

The deamination rule is simply:

```
'deamination' N(i~C!_) -> N(i~U) @ 'kdeam'
```

Note that the above rule is not atomic: we consider that the base pair is broken at the same time deamination occurs.

3.2 3D diffusion and sliding

We want to represent 3D diffusion and 2D sliding of UDG on DNA. 3D diffusion is going to be modelled by simple binding and unbinding of UDG to non specific nucleotides:

```
'binding' UDG(dbd),N(o) <-> UDG(dbd!1),N(o!1) @ 'kon'/'V','koff'
```

where `'kon'` is the constant association rate of UDG and DNA, `'V'` is the volume in which our DNA strands floats (for instance the cell cytoplasm) and `'koff'` is the dissociation rate.

Question 1: If we add no further rule, what would be the amount of UDG bound to DNA at steady state?
Answer: steady state will be reached whenever the probability of a binding event is equal to the probability of a dissociation, ie:

$$U_{free} * N * k_{on}/V = k_{off} * U_{bound}$$

where U_{free} and U_{bound} are respectively the amount of free UDG and bound UDG, and N is the number of nucleotides in the system (not the number of base pairs since UDG can bind on both Crick and Watson strands).

Remark that even with a very small k_{on}, most UDG will be bound because N is big.

We know add the rules that model 2D diffusion of UDG on DNA:

```
'slide35'  UDG(dbd!1),N(o!1,e3!2),N(o,e5!2) <->
           UDG(dbd!1),N(o,e3!2),N(o!1,e5!2) @ 'kslide','kslide'
```

Note that this rule is reversible and hence models diffusion in both 3'5' and 5'3' directions.

Question 2: Suppose that UDG lands on a particular spot on DNA, and suppose it slides during n steps. What is the average distance the enzyme will cover? Suppose that a mismatch is located k bases away from its landing spot. How many sliding steps would it take to reach it (assuming it doesn't unbind), in average.
Answer: The question is a trick and requires basic knowledge about random walks and ergodic processes to be answered. In a nutshell, the average distance the enzyme will cover is obviously 0 since the enzyme has an equal probablity to move to the left or to the right. However the random walk being ergodic, given enough time the enzyme will visit the whole strand. It takes in average n steps to visit a nucleotide that is $\sqrt{n}$ bases away. So the mismatch can be found in average in k^2 steps.

3.3 Repair steps

Whenever UDG finds a U/G mismatch it may remove the faulty U and leaves a apurinic site in its stead:

```
'U removal' UDG(dbd!1),N(o!1,i~U) -> UDG(dbd),N(o,i~AP) @ 'krepair'
```

Question 1: Which other rules form a critical pair with the above one?
Answer: this rule will compete with both directions of the `'slide35'` and with the unbinding of UDG from DNA.

Finally the Base Excision Repair (BER) pathway enzymes (not modelled here) will put a cytosine in the apurinic site and form back the C:G base pair. We simply assume here that the repair is done spontaneously.

The difficulty here is that in breaking the bond connecting both `i` sites of the paired nucleotides, we have lost the information about which nucleotide was opposite to the newly repaired one. To retrieve the information we need to exhibit a "proof" in the left hand side of the rule that the G base is indeed opposite to the repaired base:

```
'BER (1)'  N(i~AP,e5!1),N(e3!1,i!2) \
           N(i~G ,e3!3),N(e5!3,i!2) -> \
           N(i~C!0,e5!1),N(e3!1,i!2) \
           N(i~G!0,e3!3),N(e5!3,i!2) @ 'kber'
```

Strictly speaking the above rule (we use indentation to suggests opposite bases) would not allow the repair of mismatches one the 5' end of the strand.

To correct this we need the complementary rule that use the 3' direction to find the opposite base:

```
'BER (2)'  N(i~AP, e3!1), N(e5!1, i!2) \
           N(i~G , e5!3), N(e3!3, i!2)  -> \
           N(i~C!0, e3!1), N(e5!1, i!2) \
           N(i~G!0, e5!3), N(e3!3, i!2) @ 'kber'
```

However one should then deal with the fact that most mismatches will match both `'BER (i)'` rules (ie. those which are not occuring on the end pairs of the strand) and hence will be repaired twice as fast, with rate $2 * k_{ber}$.

Question 1: It seems one is forced to live with an approximation here: either neglecting the repair of a mismatch occuring on the 5' end and keep only the `'BER (1)'` rule, or have the rate of repair of mismatches occurring on the 3' and 5' end be slower than the other mismatches. Is it a bad approximation? Can you compute the likelihood that a mismatch occurs exactly on the 3' or 5' end?
Answer: Knowing that a deamination occurs, the probability that it concerns the 3' or the 5' end is the proba that the 3' end is a G:C pair and that a deamination occurrs here, plus the proba that the 5' end is a G:C pair and that a deamination occurs there, ie: $2 * (0.4/N_C)$.

Question 2: Can you suggest an alternative representation of mismatches which would not induce any approximation in the repair rate?
Answer: Instead of deleting the bond in the `'deamination'` rule, one could just maintain it and consider that it is only a symbolic link functionning as a pointer to the opposite strand. However the observable `N(i!_)` would no longer enumerate the correct base pairs, and one would be forced to check the internal state as well. Other solutions are possible of course.

4 Simulations

References

[1] Vincent Danos, Jérome Féret, Walter Fontana, and Jean Krivine. Scalable simulation of cellular signaling networks. In *Proc. APLAS'07*, volume 4807 of *LNCS*, pages 139–157, 2007.

[2] Christoffel Dinant, Martijn S. Luijsterburg, Thomas Hfer, Gesa von Bornstaedt, Wim Vermeulen, Adriaan B. Houtsmuller, and Roel van

Driel. Assembly of multiprotein complexes that control genome function. *The journal of cell biology*, 185(1):21–6, 2009.

[3] Hans E Krokan, Finn Drabløs, and Geir Slupphaug. Uracil in DNA - occurrence, consequences and repair. *Oncogene*, 21:8935–48, 2002.