DNA data storage: A generative tool for Motif-based DNA storage

Samira Brunmayr, Omer S. Sella, and Thomas Heinis, *Imperial College London*

https://www.usenix.org/conference/fast25/presentation/brunmayr

This paper is included in the Proceedings of the 23rd USENIX Conference on File and Storage Technologies.

February 25–27, 2025 • Santa Clara, CA, USA

ISBN 978-1-939133-45-8

Open access to the Proceedings of the 23rd USENIX Conference on File and Storage Technologies is sponsored by

DNA data storage: A generative tool for Motif-based DNA storage

Samira Brunmayr
Imperial College London

Omer S. Sella
Imperial College London

Thomas Heinis
Imperial College London

Abstract

DNA possesses extremely high information density and durability. In order to become a commercially viable medium such as magnetic tape and hard disk drives, the cost per bit has to decrease considerably, while the write bandwidth needs to increase. Both are governed by DNA synthesis, the process of writing data to DNA, which is currently very expensive and slow. Assembly of DNA strands from *motifs*, i.e. short DNA sequences, is an economical and faster way of representing data using DNA. Each *motif* carries a letter in an alphabet. Trading the quaternary alphabet {A,C,T,G} for longer fragments, namely motifs, allows an increase in write bandwidth and reduces cost. The success of the underlying chemistry, specifically with the assembly of motifs into polymers that faithfully represent the source binary data, is sensitive to the formation of secondary structures and the correct annealing of the motifs in a unique way. In this work, we develop a mathematical framework and a method to generate a set of motifs that agree with a predefined set of constraints regardless of the order they are combined. The set of constraints can also be easily adjusted to align with technological advances and their evolving requirements. We show that our approach generates motifs that always conform to the constraints in a more efficient manner than previous works and randomly generated motifs.

1 Introduction

Deoxyribonucleic acid (DNA) has the potential to store large amounts of digital data; up to 10^{18} bytes per mm^3. Furthermore, it has a half-life of 500 years [2], which is significantly higher than Hard Disk Drives (HDDs) and magnetic tape. In addition, DNA will always be relevant to humans, and so sequencing methods, or in this case, a read head, will always be available. These properties make DNA an excellent candidate for long-term archival storage of digital data.

Representing data at the nucleotide level, where each nucleotide corresponds to a pair of bits, albeit appealing from an information density point of view, relies on DNA synthesis. Synthesis, which refers to a variety of chemical or enzymatic processes, is at an acceptable price point for life science applications, but still too slow and costly for DNA data storage. The cost of writing one TB of data using state-of-the-art DNA synthesis [1, 20] is higher than 400 million USD, assuming a coding density of 2 bits per base. Conventional synthesis of DNA strands is thus not a viable option for commercialisation of DNA as a data-carrying medium. A novel approach [26] suggests the use of predetermined strings of nucleotides, also known as motifs [10], to form an alphabet, which in turn encodes data bits. Concatenation of symbols in the alphabet then corresponds to the concatenation of motifs using bridged oligonucleotide assembly, and results in a DNA strand that can be sequenced using one of several standard methods. This approach leads to a cheaper writing process since the motifs can be produced in advance on a large scale using Polymerase Chain Reaction (PCR) which is very cost-effective. Costly DNA synthesis is thus only used once to start a pool of motifs and is no longer required in writing data to DNA. To carry information, motifs contain a payload subsequence P, corresponding to a letter in an alphabet. To allow bridging assembly, this payload is sandwiched by one or two keys, S_0 and S_1, as shown in Figure 1 (top left). To combine motifs together, additional single-stranded DNA sequences, called bridges, that depend only on the keys, i.e., their reverse complements S_1' and S_2' are used, as further depicted in Figure 1. Reverse complementarity of S_1 and S_2 to S_1' and S_2' leads to annealing, which is followed by enzymatic ligation. The final step is the addition of missing bases to make double-stranded DNA, through Polymerase Chain Reaction (PCR), also shown in Figure 1.

If multiple distinct keys are used in the design, then multi-

This work is funded by DNAMIC (grant 101115389) and NEO (grant 101115317).

The authors are with the Department of Computing, Imperial College London, SW7 2AZ London, UK (email: samira.brunmayr18@imperial.ac.uk; o.sella@imperial.ac.uk; t.heinis@imperial.ac.uk).

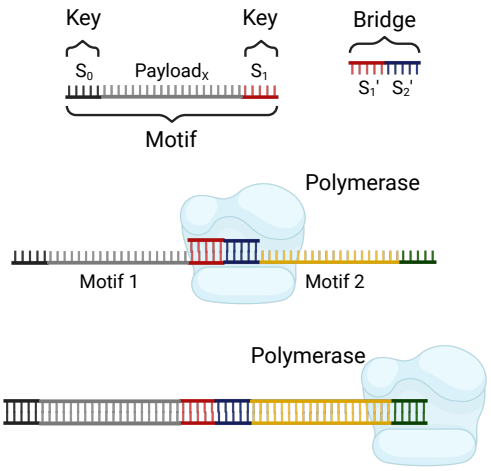

Figure 1: A payload sandwiched between two keys comprises a complete motif (top left). A bridge (top right) comprised of the concatenation of the reverse complements of $S1$ and $S2$. Motif 1, which ends with key $S1$, and Motif 2, which starts with key $S2$, anneal to the bridge (middle). Polymerase chain reaction follows in completing missing nucleotides (bottom).

ple motifs can be joined in one reaction, thereby effectively parallelizing, and thus accelerating the write process.
Encoding of data into motifs (rather than nucleotides) turns the problem of writing data onto DNA into a constrained combinatorial problem of designing keys and payloads. Specifically, a good set of motifs maintains the following properties:

1. Payloads should be uniquely and distinctly recognizable when the data-carrying DNA strand is sequenced
2. Keys should only anneal to designated reverse complemented bridges

In addition, as with most approaches of storing data on DNA, certain constraints on the resulting sequence should be met in order to ensure successful storage and reading of data:

1. GC-content limitations, i.e., the ratio of G and C nucleotide pairs should be bounded from above and below to ensure the double strand can be separated into single strands through denaturation such that it can be sequenced
2. Reducing the chance of secondary substructure formation, i.e., avoiding self-complementarity which through self-annealing can lead to hairpins and other secondary structures which render sequencing impossible
3. Avoiding homopolymers, i.e., avoiding the repetition of the same nucleotide for longer stretches
4. Elimination of reserved words and restriction sites which, if not avoided, may lead to unpredictable behaviour

It might be tempting to generate keys and payloads randomly. We show in Section 5 that this is unlikely to work. We therefore propose

1. A mathematical formulation of construction as a Markov Decision Process (MDP) in Section 2, along with a stochastic tool to generate motifs, i.e., keys and payloads, that satisfy the aforementioned biological constraints, as well as
2. A validation tool to check that designed motifs satisfy the biological constraints regardless of which motifs are joined and no matter the order in which they are joined.

As we show in Section 5, our generative tool only generates motifs that satisfy the constraints. We made the code for the generative tool, as well as the generated set of motifs used in this work in the following link: https://zenodo.org/records/12575601 and the validation tool can be found under https://zenodo.org/records/12575387.

2 Mathematical Formulation

We first assume that the set of keys, $\mathcal{K}$, is given, and proceed to construct a set of payloads $\mathcal{P}$ by induction starting from the empty set. To simplify the construction, all keys are assumed to have the same length L_K and all payloads are assumed to have the same length L_P.

2.1 Formulation as a Markov Decision Process

A partial payload is constructed in steps, extending it by one nucleotide at each step. At each step, a reward is assigned to each nucleotide from the set $\{A,T,C,G\}$ of possible extensions. The higher the reward, the more likely it is that a payload containing the corresponding base in that position would conform to all constraints. By normalizing the rewards, we obtain a categorical distribution over the set $\{A,T,C,G\}$ of possible extensions. A stochastic algorithm then chooses the next nucleotide in the sequence based on this distribution. The higher the reward for a nucleotide, the higher the probability it is selected.
Once the number of nucleotides selected reaches the designed length of the payload, L_P, i.e., the payload reaches full length, the payload is no longer partial and is committed by the algorithm to the growing set of payloads $\mathcal{P}$. Construction of a new payload is then attempted the same way, accounting for the new pool of payloads and starting from the empty payload, until no more payloads that conform to the constraints can be constructed.

2.2 An Integrated Reward Function

The reward, p_i for choosing an action a_i, i.e., extending the current sequence by appending $i \in \{A,C,T,G\}$ to it, is a

weighted sum of individual rewards corresponding to the various constraints that are to be avoided in this optimization problem. We turn to define for each base $i \in \{A,C,T,G\}$ and each constraint, x, a non-positive number which we will refer to as log score: $ls(i,x)$. We could then apply positive weights $w_x > 0$ to each log score $ls(i,x)$ and sum them across constraints for each base:

$$ls(i) = \sum_{x \in \text{constraints}} w_x \times ls(i,x) \tag{1}$$

Since $w_x > 0$ and $ls(i,x) < 0$ we have that the sum in Eq. 1 is non-positive as well, and we can then define the reward for choosing base i as:

$$p_i = \frac{e^{ls(i)}}{\sum_{j \in \{A,T,C,G\}} e^{ls(j)}} \tag{2}$$

And assure that $0 \leq e^{ls(i)} \leq 1$.
Since p_i is inversly proportional to $ls(i,x)$, we define $ls(i,x)$ as a function of x such that:

1. $ls(i,x) \leq 0$
2. $ls(i,x)$ is monotonically increasing in "violation of constraint x".

Note that there is an abuse of notation and an implicit assumption here, as we use x to note the type of constraint, but also some numeric quantity relating to it. The following details of the individual reward functions will make it clear that this assumption holds.

2.3 Homopolymer Log Score

Let p_0 be the partial payload currently being constructed, and let P be the set of payloads already committed. We consider the length of the longest homopolymer containing the last base i, appended to p_0, that can be found within any combination of p_0 with itself or any payload in P and denote it by $homLen(i)$ and depicted in Figure 2.

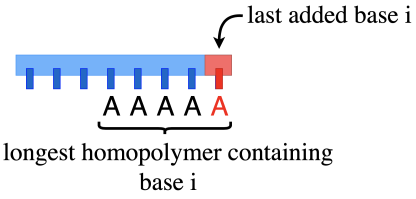

Figure 2: Illustration of a sequence being constructed where the base Adenine (A) has been appended (in red). In this example, $homLen(A) = 5$.

Since we are trying to avoid homopolymers as they cause insertion, deletion and substitution errors in a sequence during PCR [22], we can define the homopolymer log score:

$$ls(homLen,i) = -(h_{hom})^{homLen(i)/maxHom} + 1 \tag{3}$$

where h_{hom} is a hyperparameter used to shape the reward for homopolymers, and $maxHom$ is the maximal allowed length of consecutively repeated bases. The shape parameter h_{hom} allows us to control the sensitivity, or rate at which the score is affected by $homLen$, as illustrated in Figure 3.

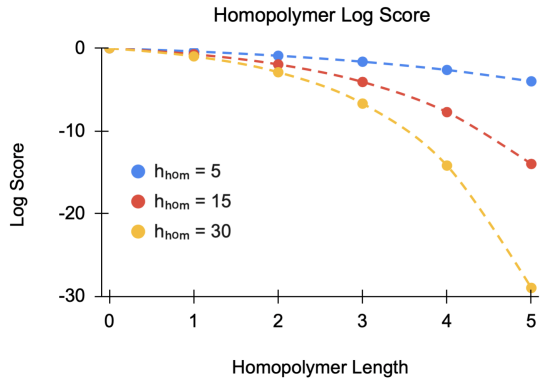

Figure 3: Homopolymer log score where the maximal allowed length of consecutively repeated bases $maxHom = 5$ and for shape hyperparameters $h_{hom} = 5$, $h_{hom} = 15$ and $h_{hom} = 30$. The closer the homopolymer length gets to the constraint boundary, the smaller the log score gets, so the smaller the probability is to select that base. The hyperparameter h_{hom} controls the slope of the function. The larger it is, the steeper the gradient of the log score will be as the homopolymer length gets closer to the boundary.

2.4 GC-Content Log Score

GC-content in a sequence is simply the percentage of bases that are either Guanine (G) or Cytosine (C). While Chemical synthesis may not be constrained by GC-content, any subsequent Chemical processes (like PCR) could be adversely affected by GC-content which is too high or low [9]. For this reason, we require a maximum GC-content $maxGC$ parameter and a minimal GC-content parameter $minGC$. Ultimately, the GC-content of **any** combination of motifs is required to exist between $minGC$ and $maxGC$. Consider the current payload, p_0, extended by $i \in \{A,C,T,G\}$, and the two quantities $curMaxGC$ and $curMinGC$ of current maximum GC and minimum GC, respectively, that could be achieved using any combination of p_0 extended by i with the set of keys and the set of payloads committed so far. Intuitively, the log score should reflect at least the worst one of the two deviations:

$$(curMaxGc - maxGC), (minGC - curMinGC) \tag{4}$$

We also need to consider the length of p_0 and the possibility of the current GC-content being adjusted by the nucleotides

added later on in the construction. As the length of p_0 becomes close to its full potential, i.e.: L_P, the log score for GC-content becomes more critical, as there will be no more steps of the construction to offset. That is why we calculate the log score for GC-content as:

$$ls(GC,i) = -\max\{0, W_{GC} \times (curMaxGc - maxGC), \\ W_{GC} \times (minGC - curMinGc))\} \quad (5)$$

where the weight W_{GC} accounts for the length of S_0:

$$W_{GC} = (\mathrm{h_{gc}})^{|p_0|/L_P} - 1 \quad (6)$$

2.5 Hairpin Log Score

Hairpins are formed when a DNA strand contains a subsequence S followed by its reverse complementary S', with some subsequence L between them as depicted in Figure 4. This may lead subsequence S to align and anneal to its reverse complementary S' forming a stem, with the subsequence L forming a loop. Hairpins, like any secondary substructures, make the accessing and sequencing of DNA sequences more difficult. The length of L, i.e., the number of nucleotides between S and S', as well as the length of the subsequence S are factors that influence the stability of the hairpin formed by S, L and S' as they increase the chances of the subsequences annealing to their reverse complements [15, 24]. The more stable the hairpins are, the more energy is required to unfold them [16]. So in the context of DNA storage, the aim is to avoid hairpins altogether or at least ensure they are unstable enough to be easily unfolded. Reduced stability is achieved by reducing the length of such subsequences S and S'. Given a range $[loopSizeMin, loopSizeMax]$, we force all hairpins with a loop size in that range to have a stem less than $maxHairpin$.

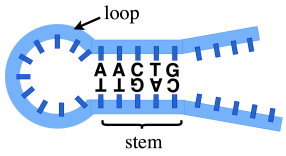

Figure 4: A hairpin is characterized by a stem and a loop.

Let s_0 be the sequence being currently constructed with added base i at the end, and let S be the set of sequences that have already been generated. Consider an index, j such that $j \in [|s_0| - maxHairpin, |s_0|]$. Let $|s_0|$ be the length of s_0, and let h be the length of the longest hairpin stem for hairpins with loop sizes between $loopSizeMin$ and $loopSizeMax$, which stem starts at position l for $l \in [|s_0| - maxHairpin, |s_0|]$. $|s_0| \leq maxHairpin$ is the longest window size of consecutive positions starting from the last added base, such that all bases have been used at least once in all motifs.

$$ls_{hairpin}(i) = \left[\sum_{h_{start}} -(\mathrm{h_{hairpin}})^{h_{start}/\mathrm{maxHairpin}} + 1\right] \quad (7)$$

where h_{start} is the length of the longest hairpin stems of all hairpins with a loop size between $loopSizeMin$ and $loopSizeMax$, and with the stem starting at position $start$ for any $h_{hairpin}$ is a shape hyperparameter.

$$start \in [|s_0| - maxHairpin, |s_0|] \quad (8)$$

However, one of our motifs design choices is that we want to be able to combine any payload with any other payload. This means that if we are trying to avoid forming a hairpin with stems s_1 and s_2, where s_2 is in the current motif we are generating and s_1 is in a different motif, we would have to avoid forming a hairpin with a stem s_1 for every single motif in the motifs set (illustrated in Figure 5).

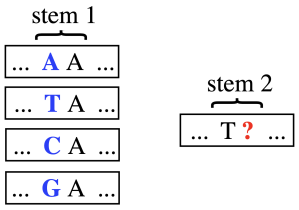

Figure 5: On the left are the previously generated motifs, and on the right is the motif that is being currently generated. In this example, a hairpin of size 2 cannot be avoided since the next base (red question mark) cannot be A, T, C nor G.

To reduce the chances of the problem mentioned before from happening, we wish to avoid a scenario in which all four bases A, T, C, G are used at the same position across all motifs within a window of $maxHairpin$ bases.
We consequently introduced a log score on the similarity of each motif:

$$ls_{similarity,\,i} = -(\mathrm{h_{similarity}})^{s/\mathrm{maxHairpin}} + 1$$

where $\mathrm{h_{similarity}}$ is the shape hyperparameter associated to the similarity between motifs. $s \leq maxHairpin$ is the longest window size of consecutive positions starting from the last added base such that all bases have been used at least once in all motifs.
Combining the two log scores mentioned above, the final hairpin log score is:

$$ls_{hairpin,\,i} = \left[\sum_{h_{start}} -(\mathrm{h_{hairpin}})^{h_{start}/\mathrm{maxHairpin}} + 1\right] + ls_{similarity,\,i} \quad (9)$$

2.6 No Key in Payload Log Score

The correct assembly of motifs is based on the annealing of keys to their reverse complement. The appearance of a key in a payload may lead to unintended annealing, which may lead to data corruption. Moreover, key-payload collisions

reduce DNA storage capacity [25]. Therefore, avoiding the appearance of keys in the payload is a good design choice. We define the log score associated to a set of keys as:

$$ls_{noKeyInPayload}(i) = -(\mathrm{h_{key}})^{seqLen/L_k} + 1 \quad (10)$$

Where L_K is the uniform length of the keys, and where, as usual, $\mathrm{h_{key}}$ is a shape hyperparameter. The variable *seqLen* is the length of the largest substring of the key that is present in the payload beginning from the last added base i to the sequence.

3 Methods

Given a set of constraints, we are able to produce a set of conforming Motifs using a stochastic tool that accepts: a set of **constraints**, a set of positive **shape hyperparameters**, and a set of positive **weight hyperparameters**.

3.1 Generation of keys

Generation of keys is similar to the generation of payloads. The states of the Markov Chains used in the generation of keys consist of:

1. the set of constraints which we wish the motifs to conform to,
2. the list of already generated distinct keys, and
3. the key being currently generated,

Each state can transition to 4 different states each containing the same elements as the previous state, except that the key being currently generated has an additional base (A, T, C, G - a different one for each state) appended to it, as described in Figure 6.
This means that given a fixed set of constraints C, the maximum number of states S_K, and maximum number of transitions T_K that the Markov Chain contains is:

$$S_K = T_K = 4^{keySize \times keyNum} \quad (11)$$

where *keySize* is the key size and *keyNum* is the maximum number of keys, both defined in C.

3.2 Generation of payloads

Similarly, the states of the Markov Chains used in the payloads generation consist of:

1. the set of constraints which we wish the motifs to conform to,
2. the list of previously generated distinct keys,
3. the set of already generated payloads, and
4. the payload being currently generated.

Each state may transition to one of four states containing the same elements as the previous state, except that the payload being currently generated has an additional base (A, T, C, G - a different one for each state) appended to it.

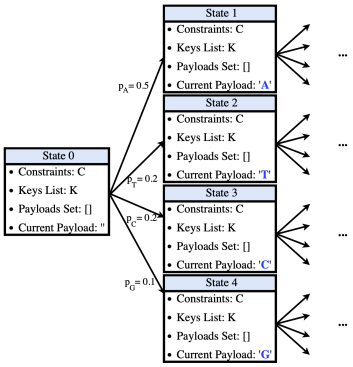

Figure 6: Example of a Markov Chain used to generate payloads with constraints C and list of keys K. The transition probabilities to choose a base A, T, C or G as the next base are p_A, p_T, p_C, p_G respectively.

This means that given a fixed set of constraints C, the maximum number of states S_P, and maximum number of transitions T_P that the Markov Chain contains is:

$$S_P = T_P = 4^{palyoadSize \times payloadNum} \quad (12)$$

where *payloadSize* is the payload size and *payloadNum* is the maximum number of payloads, both defined in C.

4 Related Work

After more than a decade of research into DNA as a digital data storage medium [8, 12], it is mainly the sequencing side that saw considerable breakthroughs. In contrast, DNA synthesis is still costly and slow, making an end-to-end, DNA-based, storage system previously envisioned, [5,6] unrealistic. This could be why some research groups turned their attention to methods that either rely on a synthesis process not suited for molecular biology [3, 28], or avoid DNA synthesis altogether [7, 27]. Our use of MDPs to generate a set of motifs generally falls into the category of Constraint Programming [19], and has some similarity to text completion methods [4, 17, 18]. The main difference, is the option to score all possible options and choose from them. Another similar task is that of constructing molecules based on physical constraints in three-dimensional space [23]. In our work, however, we consider a relatively small set of constraints, and reward has to be calculated for only four bases. If the calculation of reward becomes too extensive, or if more than four bases are considered, it may become more sensible to train a model to generate Motifs. For a full survey on DNA data storage we refer the reader to Heins et al. [14].

5 Validation and evaluation

5.1 Pass / fail validation tool

Independent of the method a set of keys and payloads is generated, we implemented a pass / fail validation tool. This tool takes as input:

- Maximal allowed length of consecutive identical bases.
- Hairpin stem maximal length.
- Hairpin loop maximal size.
- Hairpin loop minimal size.
- GC content maximal and minimal bounds.
- A **list of distinct keys** with a fixed length
- A **set of payloads** with a fixed length

A motif M is said to violate a constraint from the above list if either:

1. Motif M violates a constraint, or
2. A combination of motif M with other motifs (possibly with itself) violates a constraint.

5.2 Evaluation Against a Single Constraint

To evaluate our work, we compared the Motif Generation Tool with other existing encoding tools, namely DNA Fountain by Erlich et al. [11], Euclid by Sella et al. [21], the shortmer combinatorial encoding scheme by Preuss et al. [17], as well as randomly generated DNA sequences. This comparison was carried out by determining the time taken to generate a motif set consisting of only 1 motif (i.e., a single payload and key), conforming to each constraint separately. This procedure was performed on a quad-core Intel processor machine and was repeated for motifs with lengths between 3 and 100 bases. The results are illustrated in Figure 7. The tools by Erlich et al. and Sella et al. allow for sequence generation, conforming to constraints related to homopolymers and GC-content. However, none of the tools takes hairpins into account. It can be observed that, even though it is faster to generate a set of motifs that conform to the constraints by randomly generating them, as the length of the motifs increases, the Motif Generation Tool outperforms random generation. The only exception to this trend is when evaluating against the GC-content constraint, since the expected GC-content of randomly generated sequences is 50%. For the tool by Preuss et al., the minimum time taken across all constraints is 377ms, while for the Euclid tool, it is above 8 minutes, which are both far greater than the maximum time taken by the Motif Generation Tool, which is 25ms. Similarly for the DNA Fountain tool, the values have also been observed to be greater than those of the Motif Generation Tool. Moreover, due to design choices, the motif sizes were limited to a minimum of 60 bases, and not all motif sizes were achievable in our chosen test range.

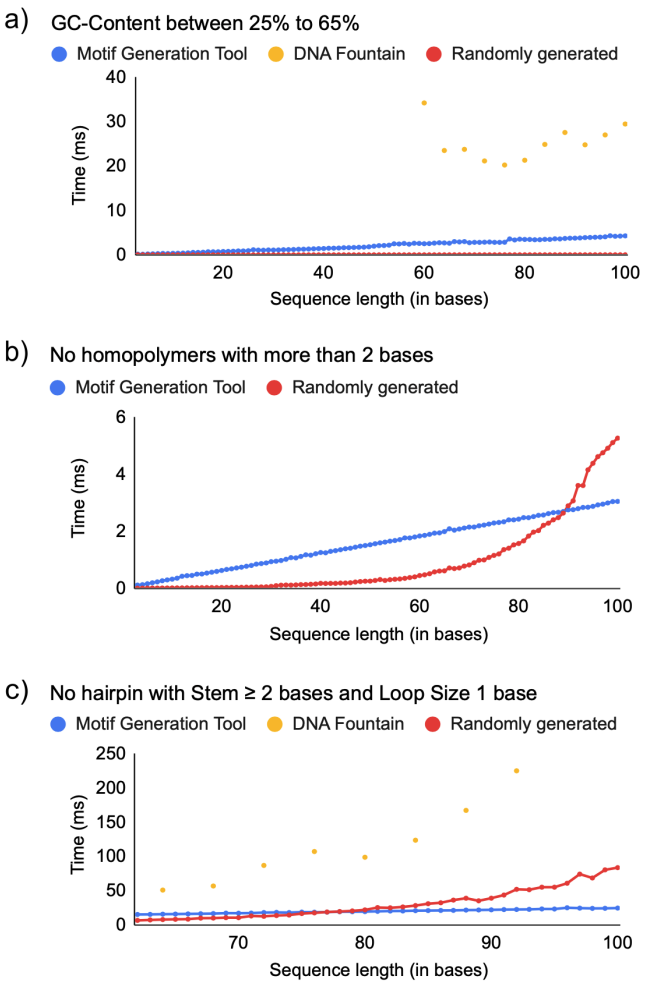

Figure 7: Time taken to generate a set of motifs consisting of 1 motif of varying lengths conforming to 3 separate constraints: a) GC-content between 25% to 65%, b) having no homopolymers with more than 2 bases, and c) having no hairpins with stem size greater than or equal to 2 bases and loop size of 1. Any times greater than 250ms are not shown. When evaluated against the homopolymer and hairpin constraints, the Motif Generation Tool always outperforms the DNA fountain [11] and the shortmer combinatorial encoding [17]. When compared against the random generation, while the Motif Generation Tool initially performs worse, its performance is better as the motif length increases. For the GC-content constraint, the Motif Generation Tool also outperforms the other tools, except for randomly generated sequences which are observed to have linear performance, as the expected GC-content of randomly generated sequences is 50%.

5.3 Evaluation Against a Set of Constraints

In this section we review the performance of the tool for biological and technological constraints shared with us by

Parameter	Value	Description
keySize	20	Size of the keys (in bp)
keyNum	8	Maximum number of keys desired to be generated
payloadSize	60	Size of the payloads (in bp)
payloadNum	15	Maximum number of payloads desired to be generated
maxHom	5	Maximum allowed length of consecutively repeated bases
maxHairpin	1	$maxHairpin$ is the maximum allowed hairpin stem length for hairpins with loop size in range [$minLoopSize$, $maxLoopSize$]
minLoopSize	6	
maxLoopSize	7	
minGC	25	Minimum allowed GC-content (in %)
maxGC	65	Maximum allowed GC-content (in %)

Table 1: Summary of parameters used to determine constraints.

a commercial DNA provider, Integrated DNA Technologies (IDT) [1] and can be found in full in prior works [20].

A summary of these constraints can be found in Table 1. Below, we will give a brief explanation as to why they were chosen.

Homopolymers of length up to 20 base pairs can be synthesised, but homopolymers of length 5 or smaller can be sequenced with better accuracy, leading us to set $maxHom = 5$.

Hairpins with loop sizes of 6 to 7 bases tend to be most stable, leading us to set $loopSizeMin = 6$ and $loopSizeMax = 7$. No data was shared on exact stem sizes which increase hairpin stability have been provided, so to avoid any hairpin, we choose $maxHairpin = 1$.

GC-Content of 25% to 65% are commonly cited leading us to set $minGc = 25$ and $maxGc = 65$.

Keys were selected to have a size of 20 base pairs. To provide at least 60% of data content from the total DNA, we set the payload uniform size to 60 since:

$$\frac{p}{2\times 20+p} \geq 0.6 \implies p \geq 60 \tag{13}$$

where p is the payload size.

This places the maximum size of a motif at $2\times 20+60 = 100$ base pairs, well within a range that could be synthesised and sequenced.

To tune hyperparameters, we used two rounds of grid search. We then used those hyperparameters, to run the Motif Generation Tool for the constraints given in Table 1 for a total of 5 minutes on a quad-core Intel processor machine, and determined the average time taken to generate a set of motifs successfully. To compare the performance of the Motif Generation Tool, we tried to generate motifs using the same constraints with DNA Fountain by Erlich et al. [11], Euclid by Sella et al. [21], shortmer combinatorial encoding scheme by Preuss et al. [17], and by randomly generating sequences. The results are depicted in Table 2. It can be seen that only the Motif Generation Tool managed to generate a set of motifs conforming to the constraints within a time frame of 5 minutes.

6 Conclusions

Using DNA Motifs to represent data, as presented by Yan et al. [26], opens a cheaper alternative to representing data

	Motif Generation Tool	DNA Fountain [11]	Euclid [21]	Preuss et al. [17]	Randomly generated
Time taken to generate a set of motifs without any constraints (seconds)	2.3×10^{-2}	1.22×10^{-1}	>5min	5.30×10^{-1}	2.1×10^{-3}
Time taken to generate a set of motifs conforming to the constraints in Table 1 (seconds)	2.54	>5min	>5min	>5min	>5min

Table 2: Tools comparison on time taken to generate a set of motifs run on a quad-core Intel processor machine. >5min means that no motif has been generated within a 5-minute time frame.

using DNA as a medium. In this work, we presented a generative tool for sets of DNA motifs that conform to a prescribed set of constraints. Our approach relies on a Markov Decision Process (MDP) [13] to produce sets of motifs, based on a parametric reward function. The reward function we presented takes into account a set of constraints on homopolymers, hairpins and GC-content. When comparing the Motif Generation Tool to existing tools, namely DNA Fountain by Erlich et al. [11], Euclid by Sella et al. [21], the shortmer combinatorial encoding scheme by Preuss et al. [17], and randomly generated sequences, it can be observed that the Motif Generation Tool outperformed the other tools as motif length and number of constraints increased. When testing the Motif Generation Tool on all constraints concurrently in Section 5.3, the tool managed to produce a set of motifs conforming to current technological and biological constraints, shared with us by a commercial DNA synthesis provider and reported by Sella et al. [20]. On the other hand, we were not able to generate such sets of motifs on any of the other previously mentioned tools. The motif generation tool we present here, is a first step in the automation of reliable DNA sequence generation for data storage, which can adapt to the technological advances which come with evolving constraints.

References

[1] Integrated dna technologies. https://eu.idtdna.com/pages. Accessed: 2025-01-17.

[2] Morten E Allentoft, Matthew Collins, David Harker, James Haile, Charlotte L Oskam, Marie L Hale, Paula F Campos, Jose A Samaniego, M Thomas P Gilbert, Eske Willerslev, et al. The half-life of dna in bone: measuring decay kinetics in 158 dated fossils. *Proceedings of the Royal Society B: Biological Sciences*, 279(1748):4724–4733, 2012.

[3] Leon Anavy, Inbal Vaknin, Orna Atar, Roee Amit, and Zohar Yakhini. Data storage in dna with fewer synthesis cycles using composite dna letters. *Nature biotechnology*, 37(10):1229–1236, 2019.

[4] Steffen Bickel, Peter Haider, and Tobias Scheffer. Learning to complete sentences. In *Machine Learning: ECML 2005: 16th European Conference on Machine Learning, Porto, Portugal, October 3-7, 2005. Proceedings 16*, pages 497–504. Springer, 2005.

[5] James Bornholt, Randolph Lopez, Douglas M Carmean, Luis Ceze, Georg Seelig, and Karin Strauss. A dna-based archival storage system. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 637–649, 2016.

[6] James Bornholt, Randolph Lopez, Douglas M Carmean, Luis Ceze, Georg Seelig, and Karin Strauss. Toward a dna-based archival storage system. *IEEE Micro*, (3):98–104, 2017.

[7] Kaikai Chen, Jinbo Zhu, Filip Boskovic, and Ulrich F Keyser. Nanopore-based dna hard drives for rewritable and secure data storage. *Nano Letters*, 20(5):3754–3760, 2020.

[8] George M Church, Yuan Gao, and Sriram Kosuri. Next-generation digital information storage in dna. *Science*, 337(6102):1628–1628, 2012.

[9] Clara Delahaye and Jacques Nicolas. Sequencing dna with nanopores: Troubles and biases. *PloS one*, 16(10):e0257521, 2021.

[10] Patrik D'haeseleer. What are dna sequence motifs? *Nature biotechnology*, 24(4):423–425, 2006.

[11] Yaniv Erlich and Dina Zielinski. Dna fountain enables a robust and efficient storage architecture. *science*, 355(6328):950–954, 2017.

[12] Nick Goldman, Paul Bertone, Siyuan Chen, Christophe Dessimoz, Emily M LeProust, Botond Sipos, and Ewan Birney. Towards practical, high-capacity, low-maintenance information storage in synthesized dna. *Nature*, 494(7435):77–80, 2013.

[13] Brian Hayes et al. First links in the markov chain. *American Scientist*, 101(2):252, 2013.

[14] Thomas Heinis, Roman Sokolovskii, and Jamie J Alnasir. Survey of information encoding techniques for dna. *ACM Computing Surveys*, 56(4):1–30, 2023.

[15] Dafa Li, Hongtao Huang, Xinxin Li, and Xiangrong Li. Hairpin formation in dna computation presents limits for large np-complete problems. *Biosystems*, 72(3):203–207, 2003.

[16] Thijs Nieuwkoop, Max Finger-Bou, John van der Oost, and Nico J Claassens. The ongoing quest to crack the genetic code for protein production. *Molecular cell*, 80(2):193–209, 2020.

[17] Inbal Preuss, Michael Rosenberg, Zohar Yakhini, and Leon Anavy. Efficient dna-based data storage using shortmer combinatorial encoding. *Scientific reports*, 14(1):7731, 2024.

[18] Nathaniel Roquet, Swapnil P Bhatia, Sarah A Flickinger, Sean Mihm, Michael W Norsworthy, Devin Leake, and Hyunjun Park. Dna-based data storage via combinatorial assembly. *bioRxiv*, pages 2021–04, 2021.

[19] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[20] Omer Sella. *Coding for emerging archival storage media*. PhD thesis, University of Cambridge, 2024.

[21] Omer S Sella, Amir Apelbaum, Thomas Heinis, Jasmine Quah, and Andrew W Moore. Dna archival storage, a bottom up approach. In *Proceedings of the 13th ACM Workshop on Hot Topics in Storage and File Systems*, pages 58–63, 2021.

[22] Deepali Shinde, Yinglei Lai, Fengzhu Sun, and Norman Arnheim. Taq dna polymerase slippage mutation rates measured by pcr and quasi-likelihood analysis:(ca/gt) n and (a/t) n microsatellites. *Nucleic acids research*, 31(3):974–980, 2003.

[23] Gregor Simm, Robert Pinsler, and Jose Miguel Hernandez-Lobato. Reinforcement learning for molecular design guided by quantum mechanics. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 8959–8969. PMLR, 13–18 Jul 2020.

[24] Alexander Vologodskii. *Biophysics of DNA*. Cambridge University Press, 2015.

[25] Yixun Wei, Bingzhe Li, and David HC Du. An encoding scheme to enlarge practical dna storage capacity by reducing primer-payload collisions. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 71–84, 2024.

[26] Yiqing Yan, Nimesh Pinnamaneni, Sachin Chalapati, Conor Crosbie, and Raja Appuswamy. Scaling logical density of dna storage with enzymatically-ligated composite motifs. *Scientific Reports*, 13(1):15978, 2023.

[27] Yiqing Yan, Nimesh Pinnamaneni, Sachin Chalapati, Conor Crosbie, and Raja Appuswamy. Scaling logical density of dna storage with enzymatically-ligated composite motifs. *Scientific Reports*, 13(1):15978, 2023.

[28] Meng Yu, Xiaohui Tang, Zhenhua Li, Weidong Wang, Shaopeng Wang, Min Li, Qiuliyang Yu, Sijia Xie, Xiaolei Zuo, and Chang Chen. High-throughput dna synthesis for data storage. *Chemical Society Reviews*, 2024.