使用阴阳编解码器系统实现实用且稳健的基于 DNA 的数据归档

张红玫

摘要

DNA 是一种很有前途的数据存储介质，因为它具有出色的耐用性和节省空间的存储能力。早期的转码算法有解码不成功、合成和测序方面有挑战等问题，因此，开发一种能够实现高信息密度，对于在实际应用中基于 DNA 的信息存储，以经济高效的方式对各种数据类型执行稳健可靠的转码的算法是很有必要的。这里，我们提出了一种名为阴阳编解码器的强大转码算法，使用两个规则将两个二进制位编码为一个核苷酸，以生成与合成和测序技术高度兼容的 DNA 序列。

关键词：DNA 数据存储；阴阳编解码器

1 引言

DNA 是生物体内一种古老而高效的信息载体。目前，它被认为具有作为替代存储介质的巨大潜力，因为标准存储介质已不能满足呈指数增长的数据归档需求。已经提出了许多使用有机分子存储数字信息的策略，由于当前的 DNA 测序技术在成本和通量方面都具有优势，因此使用 DNA 分子存储数字信息仍然是最广为接受的策略。

使用基本的转码规则会在 DNA 序列中产生一些特定模式，从而导致合成和测序方面的挑战[1]。例如，长于 5nt 的单核苷酸重复（同聚物）可能会在合成或测序过程中引入更高的错误率 1。同时，由于互补碱基配对的性质（A 与 T 配对，G 与 C 配对），DNA 分子可能形成发夹或拓扑假结（即二级结构）等结构，这可以通过计算自由能来预测它的顺序。据报道，具有稳定二级结构的 DNA 序列可能不利于测序或使用 PCR 随机访问和备份存储信息[2]。此外，GC 含量 <40% 或 >60% 的 DNA 序列通常难以合成。因此，均聚物的长度（以 nt 为单位）、二级结构（由以 kJ mol−1 为单位的计算自由能表示)和 GC 含量 (%) 是评估编码方案兼容性的三个主要参数。

先前关于转码算法开发的研究试图提高生成的 DNA 序列的兼容性。例如，DNA Fountain 算法采用 Luby 变换代码，通过引入低冗余以及对均聚物长度和 GC 含量的筛选约束来提高信息保真度，同时保持 1.57 位 nt-1 的信息密度[3]。然而，主要缺点是由于 Luby 变换代码的基本问题，在处理特定二进制特征时存在解码不成功的风险。这种方法依赖于引入足够的逻辑冗余，即在编码级别，用于容错以确保成功解码。减少逻辑冗余可能导致解码失败的概率很高，但过多的逻辑冗余会降低信息密度并显着增加合成成本 25. 此外，使用这些早期算法的特定二元模式也可能产生不合适的 DNA 序列，具有极端 GC 含量或长均聚物。

因此开发一种能够实现高信息密度的编码算法，但更重要的是，对于在实际应用中开发基于 DNA 的信息存储,有必要以经济高效的方式对各种数据类型执行稳健可靠的转码算法。阴阳编解码器 (YYC) 编码算法，其灵感来自中国传统的阴阳概念，代表两种不同但互补且相互依赖的规则，YYC 的优点是

结合了阴阳规则，最终产生了 1536 种编码方案，可以适应不同的数据类型，可以有效地消除长均聚物序列的产生，同时将生成的 DNA 序列的 GC 含量保持在可接受的水平内。

2 相关工作

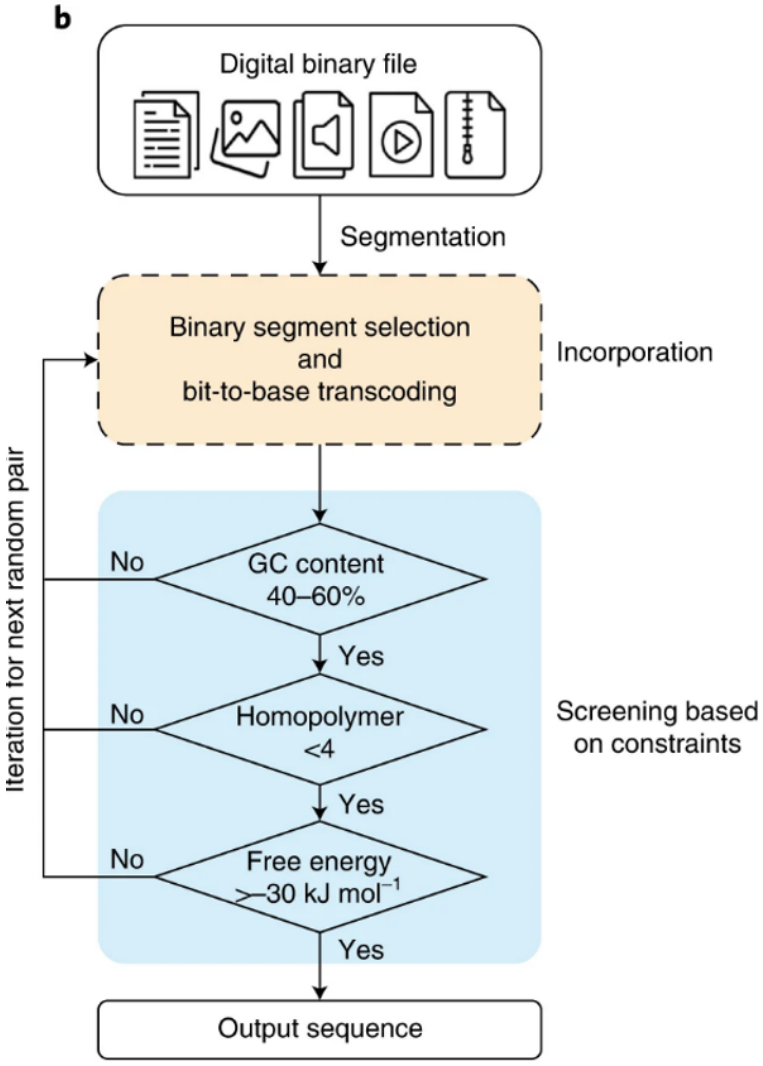


图 1: YYC 编码流水线流程图

YYC 编码流水线流程如图 1 所示，首先要对二进制数据文件进行分段，然后进行合并，合并指的是随机选择两个二进制片段并转换为碱基序列。YYC 算法筛选过程的生成的 GC 含量 >60% 或 >40%、携带 >6 聚体均聚物区域或预测二级结构 <-30kcal mol-1 的 DNA 序列（通常为 200 nt）将被拒绝。然后，将执行新一轮的片段配对以重复筛选过程，直到生成的 DNA 序列满足所有筛选标准。

3 本文方法

3.1 本文方法概述

YYC 算法的一般原理是将两个独立的编码规则，称为“yin”和“yang”，合并到一个 DNA 序列中（称为“incorporation”），从而将两个位压缩为一个核苷酸，如图 2所示：

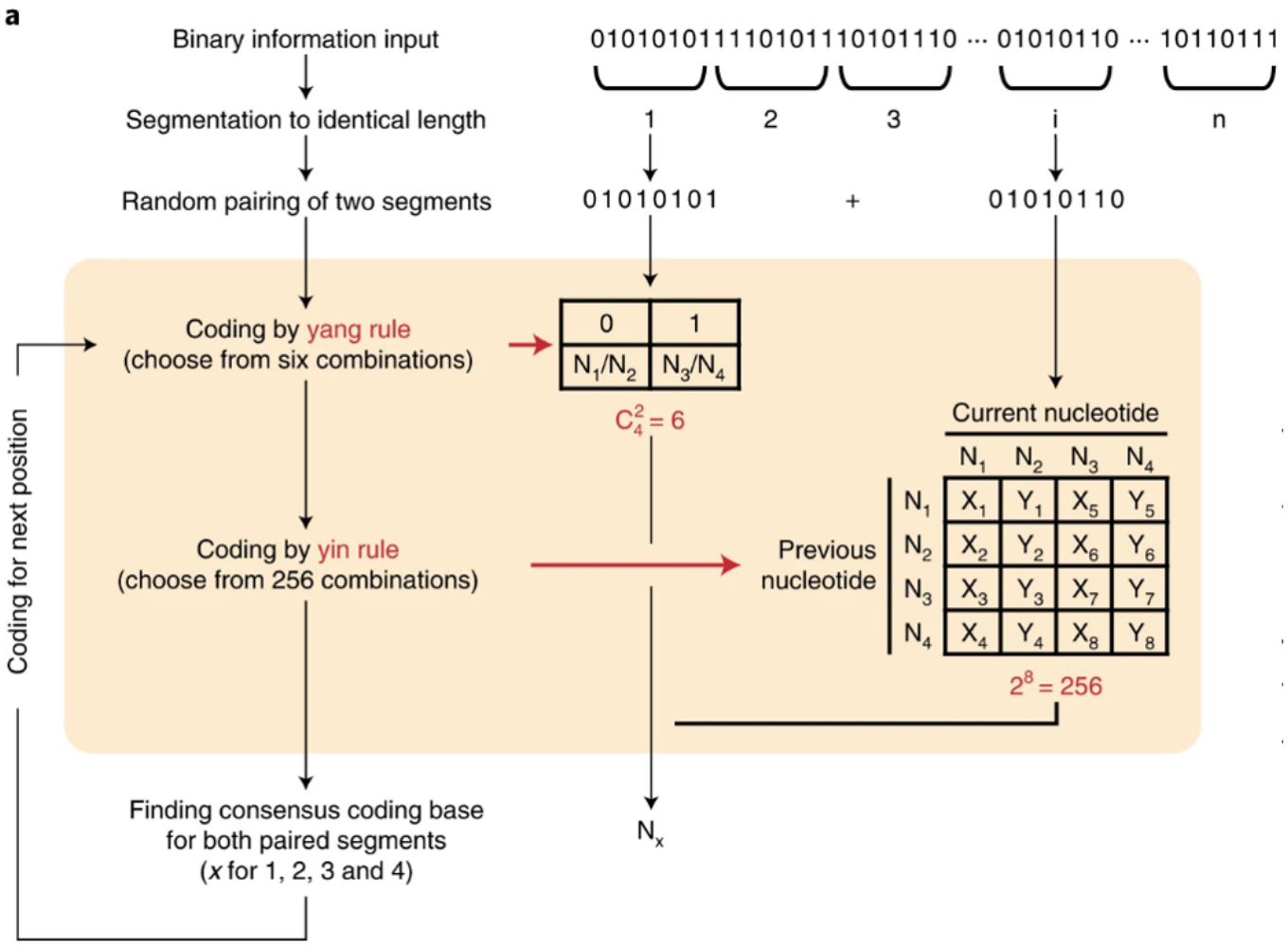
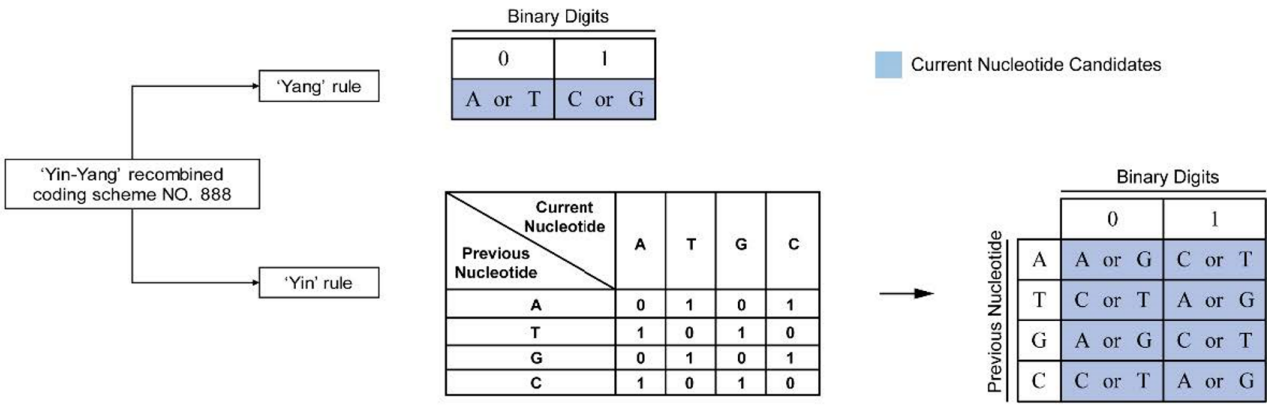


图 2: YYC 的 bit-to-base 转码过程

这里，用 N1、N2、N3、N4 分别代表 A、T、C、G 四种核酸。X_j 和 Y_j 表示不同的二进制数字 0 和 1。当 j 为 1 到 8 中的整数时，$X_j + Y_j = 1$，$X_j \times Y_j = 0$（即 8 个独立的 X 和 Y 集合, 其中 X_j/Y_j 为 1/0 或 0/1)。对于一个选定的组合编码方案，输出 DNA 序列是通过合并两个相同长度的二进制片段生成的。第一步，应用 yang 规则生成六种不同的编码组合。那么在 yin 规则中，N1 和 N2 映射到不同的二进制数位，而 N3 和 N4 也映射到独立于 N1 和 N2 的不同二进制数位，总共有 256 种不同的编码组合。在一个位置应用阴阳规则将产生一个且只有一个共有核苷酸。同时，根据前一个核苷酸的四个不同选项，两组（N1/N2 和 N3/N4）也有独立的映射到 0 和 1 的选项。因此，合并的阴阳规则总共提供了 1536(6 × 256) 转码方案组合来编码二进制序列。

3.2 YYC 编码举例

a



b

A step-by-step transcoding demonstration using coding scheme No. 888

Input Signals a = 10110011 b = 01011101

Transcoding According to Rules

	Previous Nucleotide	Binary Digits	Rule	Candidates	Local Nucleotide	Oligo Sequence
Step 1	A virtual	a1 = 1 b1 = 0	'Yang' 'Yin'	C or G A or G	G	G
Step 2	G	a2 = 0 b2 = 1	'Yang' 'Yin'	A or T C or T	T	GT
. . .						
Step 8	G	a8 = 1 b8 = 1	'Yang' 'Yin'	C or G C or T	C	GTCGTAGC

Result Sequence: GTCGTAGC

图 3: 使用 YYC 的 1536 编码方案之一的代码转换说明举例

如图 3中的 a 所示，图中' a1 '，b1 ' 表示' a ' 和' b ' 段中第一个位置的二进制数字，以此类推。虚拟碱基 A 意味着该碱基仅用于确定转码第一次运行时的输出碱基，并且不会出现在结果序列中。阳规则规定 [A，T] 代表二进制数字 0，而 [G，C] 代表二进制数字 1。同时，阴规则规定局部核苷酸由前面的核苷酸（或“支持核苷酸”）和相应的二进制数字决定。这两个规则分别应用于两个独立的二进制片段并转码成一个唯一的 DNA 序列，而解码则以相反的顺序进行。例如，给定输入信号分别由“10110011”和“01011101”的“a”和“b”组成，转码方案将从每个片段中的第一个核苷酸开始。根据阳律，'a' 中的'1' 提供了两个选项 [C，G]。使用位置 0 的预定义虚拟核苷酸作为“A”，阴规则和“b”的“0”也提供两个选项 [A，G]。因此，这两个集合的交集生成唯一的基数 [G] 转码这两个段中的第一个二进制数字。类似地，这两个片段的其余部分可以转换为独特的核苷酸序列。

4 复现细节

4.1 与已有开源代码对比

头文件代码如下：

$[support_base, rule1, rule2] = [\text{"}A\text{"}, [0,1,0,1], [[1,1,0,0],[1,0,0,1],[1,1,0,0],[1,1,0,0]]$ # 原代码

$[support\_base, rule1, rule2] = [\text{"}A\text{"}, [0,1,0,1], [[1,0,0,1], [1,0,0,1], [1,1,0,0], [0,1,1,0]]]$

$tool = scheme.YYC(support\_bases = support\_base, base\_reference = rule1, current\_code\_matrix = rule2, search\_count = 100, max\_homopolymer = 4, max\_content = 0.6)$

pipeline.encode(method=tool, input_path=read_file_path, output_path=dna_path, model_path=model_path, need_index=True, need_log=True)

del tool

pipeline.decode(model_path=model_path, input_path=dna_path, output_path=write_file_path, has_index=True need_log=True)

4.1.1 把二进制文件编码 DNA 序列

```
def encode(self, matrix, size, need_log=False):
    # 从总数据中分离“好”和“坏”数据，并将索引和数据拼接为列表。
    good_data_set, bad_data_set = self._divide_library(matrix, need_log)
    # 匹配“好”数据和“坏”数据，以确保整体数据更好。如果只剩下“好”或“坏”数据，它们将被选择配对。
    data_set = self._pairing(good_data_set, bad_data_set, need_log)
    # 二维数据集合成 dna 序列
    dna_sequences = self._synthesis_sequences(data_set, need_log)
return dna_sequences
```

4.1.2 读取 DNA 序列还原数据文件

```
def decode(method=None, model_path=None, input_path=None,output_path=None,verify=None, has_index=T need_log=False):
    从文件中读取 DNA 序列集。
    dna_sequences = data_handle.read_dna_file(input_path, need_log)
    # 解码 DNA 序列到二进制文件的数据，一个 DNA 序列 <-> 双行二进制。
    output_matrix, size = method.decode(dna_sequences, need_log)
    if has_index:
    # 将数据与二进制字符串中的索引分开。
       indexes, data_set = index_operator.divide_all(output_matrix, need_log)
    # 按索引顺序恢复数据。
       output_matrix = index_operator.sort_order(indexes, data_set, need_log)
    # 写入二进制矩阵文档。
    data_handle.write_all_from_binary(output_path, output_matrix, size, need_log)
```

4.2 实验环境

所有编码、解码和错误分析实验均在 Windows 系统中的 Pycharm 软件上执行，该环境具有 16 GB 随机存取存储器的 i7 中央处理器，使用 Python 3.7.0。使用 YYC 算法对一个 136.65 KB 的 jpg 图片进行编码，约束设置如下：GC 含量为 40-60%，最大允许均聚物长度为 4。寡核苷酸长度都设置为 130 个碱基，带有用于数据检索的索引或种子，并且没有纠错码。

4.3 创新点

代码测试对图片文件进行了转码，通过修改规则，使其满足相关条件，使用不同的规则对不同的图片进行转码，结果也能成功，验证了算法的鲁棒性。

5 实验结果分析

使用 YYC 算法对一个 136.65 KB 的 jpg 图片进行编码，把生成的 DNA 序列保存到本地，然后读取 DNA 序列文件，通过解码算法转换为 jpg 图片并保存到本地, 结果如下两图所示：

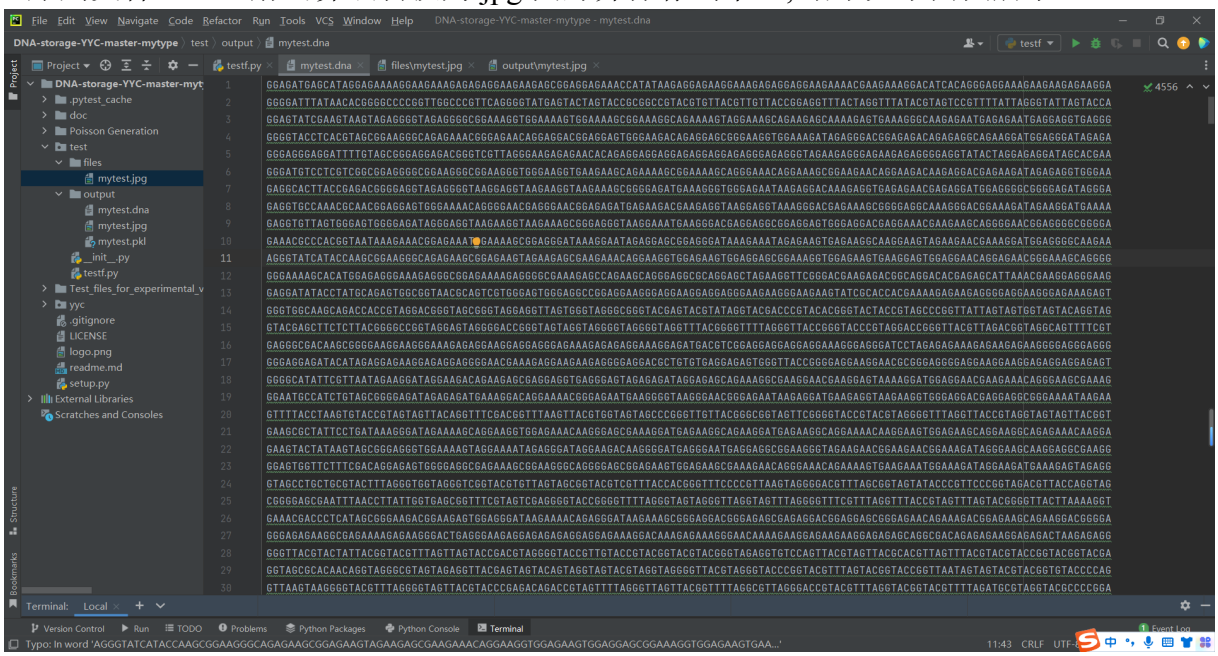


图 4: 编码成的 DNA 序列

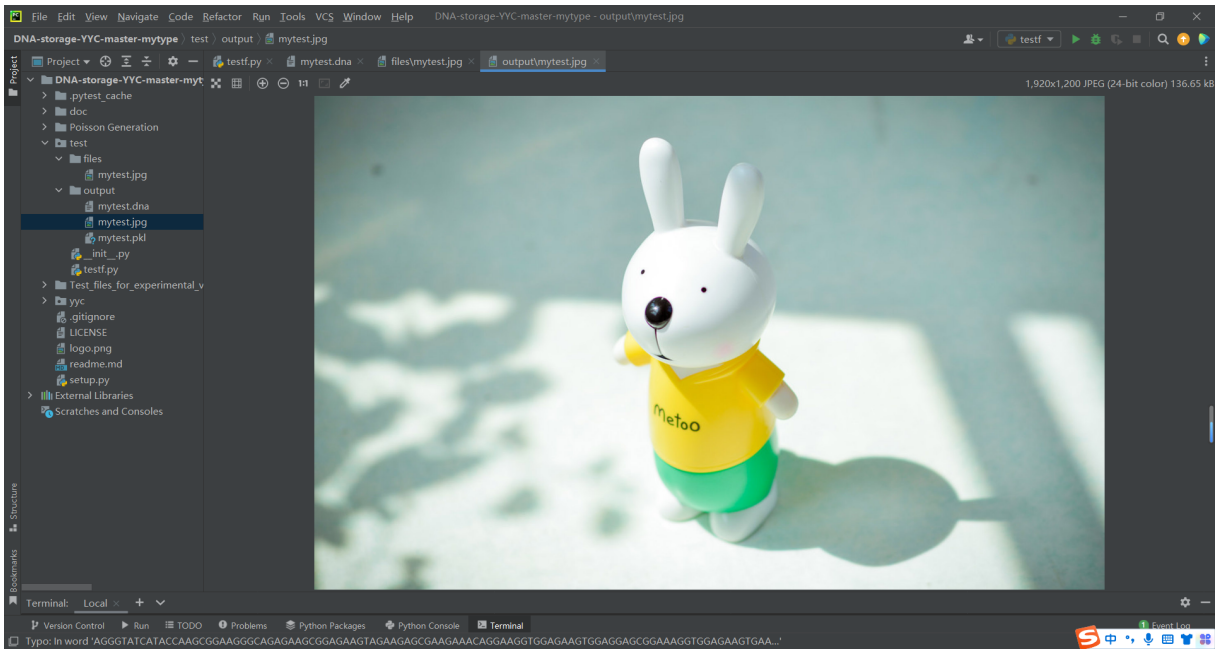


图 5: 解码得到的数据文件

6 总结与展望

通过复现这篇文献，我第一次接触到了 DNA 存储的现状，了解到了如何通过 DNA 来存储数据，以及 DNA 存储数据包括的步骤：编码、合成、存储、测序、解码。YYC 转码算法具有多种优势。首先，与其他早期努力相比，它成功地平衡了 DNA 数据存储的高稳健性、兼容性和相当大的信息密度。其次，YYC 为单个文件的转码提供了合并多种编码方案的机会，从而为安全数据归档提供了一种替代策略。在复现代码的过程中，我理解了如何把读取出来的二进制数据，通过 YYC 算法转换为限定的条件下的 DNA 序列，并通过解码算法还原数据。目前仍然需要克服的难题是 DNA 合成的成本过高，可以在未来通过使用具有高逐步效率、通量和保真度的 DNA 合成技术来解决。

参考文献

[1] TABATABAEI YAZDI S, YUAN Y, MA J, et al. A rewritable, random-access DNA-based storage system [J]. Scientific reports, 2015, 5(1): 1-10.

[2] KIELECZAWA J. Fundamentals of sequencing of difficult templates—an overview[J]. Journal of biomolecular techniques: JBT, 2006, 17(3): 207.

[3] KOCH J, GANTENBEIN S, MASANIA K, et al. A DNA-of-things storage architecture to create materials with embedded memory[J]. Nature biotechnology, 2020, 38(1): 39-43.