Journal of Applied Mathematics & Bioinformatics, vol.3, no.3, 2013, 1-40
ISSN: 1792-6602 (print), 1792-6939 (online)
Scienpress Ltd, 2013

A necklace algorithm to determine the growth function of trinucleotide circular codes

Matthieu Herrmann[1], **Christian J. Michel**[2,3] **and Benoît Zugmeyer**[4]

Abstract

Circular codes are mathematical objects studied in combinatorics - theoretical computer science, and theoretical biology. So far, there is no close formulas allowing to determine the growth function (number and list) of circular codes. This combinatorial problem can only be solved by an algorithmic approach. We propose a new algorithm based on a necklace proposition to determine the growth function of trinucleotide circular codes, a trinucleotide being a word of 3 letters on a 4-letter alphabet. This necklace algorithm, unique in its class, can be extended in future to the analysis of codes, e.g. circular codes, containing words greater than 3 letters and also over larger alphabets.

[1] Equipe de Bioinformatique Théorique, ICube, Université de Strasbourg, CNRS, 300 Boulevard Sébastien Brant, 67400 Illkirch, France.
BioEmergences, INAF - CNRS, ISC - Paris Ile de France, Avenue de la Terrasse, 91190 Gif Sur Yvette, France.
[2] Equipe de Bioinformatique Théorique, ICube, Université de Strasbourg, CNRS, 300 Boulevard Sébastien Brant, 67400 Illkirch, France.
[3] Corresponding Author.
[4] Equipe de Bioinformatique Théorique, ICube, Université de Strasbourg, CNRS, 300 Boulevard Sébastien Brant, 67400 Illkirch, France.

Article Info: *Received* : April 16, 2013. *Revised* : June 4, 2013
Published online : September 15, 2013

Mathematics Subject Classification: 94A45; 68Q99
Keywords: circular code; trinucleotide; necklace algorithm; combinatorial algorithm

1 Introduction

Comma free codes, a very particular case of circular codes, have been studied for a long time, e.g. [7, 10, 11]. After the discovery of a circular code in genes with strong mathematical properties [1], circular codes are mathematical objects studied in combinatorics - theoretical computer science, e.g. [3, 2, 23, 27, 20, 21, 24, 25, 18, 26, 19, 5, 6, 22], and theoretical biology, e.g. [15, 29, 14, 8, 9, 17, 13, 12, 28].

So far, there is no close formulas to determine the growth functions (numbers and sets of words) of trinucleotide circular codes. The only way to solve this combinatorial problem is the algorithmic approach. The determination of growth functions of small classes of trinucleotide circular codes, e.g. the $99,320$ self-complementary trinucleotide circular codes [27] and the ≈ 559 million trinucleotide comma-free codes [20], can be obtained by using the classical flower automaton algorithm [4]. The identification of the growth function of the ≈ 116 billion (10^9) trinucleotide circular codes [18] among $1,100$ billion potential codes has required the developement of a new algorithm based on a necklace proposition. Indeed, this problem has a computational complexity with an order of magnitude significantly higher, more than 200 times, than the determination of growth functions in the two previous cases. The necklace algorithm, called NA, allows to solve such a combinatorial problem. It involves several computer techniques based on a generated trinucleotide matrix, branch pruning, parallelization and different implementation hints. The definitions of the necklace algorithm NA are first presented in the one dimension case associated to the trinucleotide lexicographical order and then extended to the two dimension case associated to the trinucleotide conjugation classes. This NA algorithm presentation allows not only to introduce the concepts progressively but also to extend it in future to the analysis of codes, e.g. circular codes, containing words greater than 3 letters and also over larger alphabets, either by using a word lexicographical order or word conjugation classes.

2 Preliminaries

The following definitions and propositions are classical for any finite set of words on any finite alphabet [4]. We recall them for trinucleotides, i.e. words of length 3 on a 4-letter alphabet. Let $\mathcal{A}_4 = \{A, C, G, T\}$ denote the genetic alphabet, lexicographically ordered by $A < C < G < T$. The set of non-empty words (resp. words) on $\mathcal{A}_4$ is denoted by $\mathcal{A}_4^+$ (resp. $\mathcal{A}_4^*$). The set of the 64 words of length 3 (trinucleotides or triletters) over $\mathcal{A}_4$ is denoted by $\mathcal{A}_4^3$.

Definition 2.1. *A set X of words in $\mathcal{A}_4^3$ is a trinucleotide code if, for each $x_1, \ldots, x_n, x'_1, \ldots, x'_m \in X$, $n, m \geq 1$, the condition $x_1 \cdots x_n = x'_1 \cdots x'_m$ implies $n = m$ and $x_i = x'_i$ for $i = 1, \ldots, n$.*

Trinucleotide codes are read on a straight line.

Definition 2.2 ([16]). *A trinucleotide code X in $\mathcal{A}_4^3$ is circular if, for each $x_1, \ldots, x_n, x'_1, \ldots, x'_m \in X$, $n, m \geq 1$, $p \in \mathcal{A}_4^*$, $s \in \mathcal{A}_4^+$, the conditions $s x_2 \cdots x_n p = x'_1 \cdots x'_m$ and $x_1 = ps$ imply $n = m$, $p = \varepsilon$ (empty word) and $x_i = x'_i$ for $i = 1, \ldots, n$.*

Trinucleotide circular codes are read on a circle.

Proposition 2.3 ([4]). *A trinucleotide circular code cannot contain a word of the form u^3 with $u \neq \varepsilon$.*

The periodic trinucleotides $\{AAA, CCC, GGG, TTT\}$ cannot be in a trinucleotide circular code. The set $\mathcal{A}_4^3$ is a trinucleotide code but not a trinucleotide circular code.

Remark 2.4. *Two trinucleotides u and v are* conjugate *if there exist two words s and t such that $u = st$ and $v = ts$.*

Proposition 2.5 ([4]). *A trinucleotide circular code cannot contain conjugate trinucleotides.*

The conjugate trinucleotides ACG and CGA cannot be in the same circular code.

Definition 2.6. *The circular* permutation map $\mathcal{P} : \mathcal{A}_4^3 \to \mathcal{A}_4^3$ *permutes circularly each trinucleotide $l_1 l_2 l_3$ as follows $\mathcal{P}(l_1 l_2 l_3) = l_2 l_3 l_1$. For example,*

$\mathcal{P}(AAC) = ACA$. *The* k*th iterate of* $\mathcal{P}$ *is denoted* $\mathcal{P}^k$. *This map on words is also naturally extended to word sets: a permuted trinucleotide set is obtained by applying the circular* permutation map $\mathcal{P}$ *to all its trinucleotides.*

Remark 2.7. *Therefore, if* u *and* v *satisfy* $\mathcal{P}^k(u) = v$ *for some* k*, then* u *and* v *are conjugate.*

Definition 2.8. *A trinucleotide circular code* X *in* $\mathcal{A}_4^3$ *is maximal if for each* $x \in \mathcal{A}_4^3, x \notin X, X \cup \{x\}$ *is not a trinucleotide circular code.*

The following lemma is very well known.

Lemma 2.9 ([4]). *For any letter* α, β, γ *and for any circular trinucleotide code* X*, then* $\alpha\alpha\alpha \notin X$ *and the set* $\{\alpha\beta\gamma, \beta\gamma\alpha, \gamma\alpha\beta\} \cap X$ *contains at most one element and exactly one when* X *has* 20 *elements.*

Remark 2.10. *The conjugation class of the trinucleotide* AAA *has only one element:* AAA *itself. Obviously, this property is also true for the trinucleotides* CCC, GGG, TTT. *Otherwise, each other trinucleotide belongs to a conjugation class having exactly three trinucleotides. Consequently, the non-periodic trinucleotides, i.e.* $\mathcal{A}_4^3 \setminus \{AAA, CCC, GGG, TTT\}$*, are partitioned into exactly* 20 *classes. Finally, any trinucleotide circular code* X *with* 20 *words is maximal.*

Remark 2.11. *The length* l *(number of words) of trinucleotide circular codes varies between 1 and 20.*

The set X of 20 trinucleotides identified in the gene populations of both eukaryotes and prokaryotes is a maximal trinucleotide circular code [1].

3 Circular code propositions

Proposition 3.1. *The number of trinucleotide circular codes of length 1 is equal to 60.*

Proof. Obvious. □

Proposition 3.2 ([1]). *The number of trinucleotide circular codes of length 20 is equal to 12,964,440.*

Proof. This number was obtained in 1996 by using the flower automaton algorithm (Table 2(d) in [1]). □

In order to compute the growth function of trinucleotide circular codes for all lengths $l = 1, \ldots, 20$, we extend the necklace definition [23].
$l_1, l_2, \ldots, l_{n-1}, l_n, \ldots$ are letters in $\mathcal{A}_4$, $d_1, d_2, \ldots, d_{n-1}, d_n, \ldots$ are diletters in $\mathcal{A}_4^2$ and n is an integer satisfying $n \geq 2$.

Definition 3.3. *Letter Diletter Continued Closed Necklaces (LDCCN): We say that the ordered sequence $l_1, d_1, l_2, d_2, \ldots, d_{n-1}, l_n, d_n, l_{n+1}$ is a necklace $(n+1)\,LDCCN$ for a subset $X \subset \mathcal{A}_4^3$ if $l_1 d_1, l_2 d_2, \ldots, l_n d_n \in X$ and $d_1 l_2, d_2 l_3, \ldots, d_{n-1} l_n, d_n l_{n+1} \in X$ and $l_1 = l_{n+1}$.*

Proposition 3.4 ([18]). *Let X be a trinucleotide circular code. The following conditions are equivalent:*

1. *X is a trinucleotide circular code.*
2. *X has no necklace $nLDCCN$ for any integer $n \in \{2, 3, 4, 5\}$.*

Proposition 3.5. *A trinucleotide code has a necklace $2LDCCN$ if and only if it has a trinucleotide in $\mathcal{L}_4^3 = \{AAA, CCC, GGG, TTT\}$ or two conjugate trinucleotides.*

Table 1 in [18] gives the number $\mathrm{Nb}\,(l)$ of trinucleotide circular codes of length l, $l = 1, \ldots, 20$. The growth function has a minimum number $\mathrm{NbMin} = 60$ at $l = 1$ and a maximum number $\mathrm{NbMax} = 23,403,485,556$ at $l = 13$.

Proposition 3.6 ([4]). *Let X and Y be two trinucleotide codes with $X \subset Y$. If X has an $nLDCCN$ (i.e. is not circular) then Y also has an $nLDCCN$ (i.e. is also not circular).*

4 Necklace algorithm NA

4.1 The necklace algorithm NA in one dimension (lexicographical order)

4.1.1 Principle of the necklace algorithm NA

We describe here a new algorithm based on the necklace concept, called NA, to compute very quickly the growth function of trinucleotide circular codes, i.e. their numbers and their lists of trinucleotides [18]. The total number NbPTCC of potential trinucleotide circular codes $PTCC$ for all lengths $l = 1, \dots, 20$ is NbPTCC $= \sum_{l=1}^{20} \binom{20}{l} \times 3^{l} \approx 1.1 \times 10^{12}$ (consequence of Proposition 3.5). Generating all the combinations of trinucleotide codes and testing them to be circular is time consuming, even for actual computers. The new algorithm NA allows to identify the growth function of trinucleotide circular codes for all lengths $l = 1, \dots, 20$ in a few hours on a standard personal computer. A naive algorithm would need several weeks on a personal computer.

The necklace algorithm NA is based on Propositions 3.4, 3.5 and 3.6:

- NA generates by construction trinucleotide codes without necklace $2LDCCN$ (Proposition 3.5). Thus, it avoids the $2LDCCN$ tests and generates only NbPTCC trinucleotide codes instead of all possible trinucleotide code combinations over $\mathcal{A}_4^3$, i.e. $\sum_{l=1}^{20} \binom{64}{l} \approx 3.4 \times 10^{16}$.
- If NA identifies a trinucleotide code X with a necklace $nLDCCN$ for a given $n \in \{3, 4\}$ then it classifies X as being not circular and avoids to analyse the larger necklaces $n'LDCCN$ for $n' > n$, $n' \in \{4, 5\}$ (Proposition 3.4).
- Trinucleotide codes are incrementally generated by increasing their lengths. A trinucleotide code X_l of length l, $l \leq 20$, is constructed after the generation of $(l-1)$ trinucleotide subcodes X_m of length $m < l$ contained in X_l, i.e. $X_1 \subset X_2 \subset \cdots \subset X_{l-1} \subset X_l$, which are circular. Indeed, if a trinucleotide code X_m has a necklace $nLDCCN$ for a given $n \in \{3, 4, 5\}$, i.e. is not circular, then any following larger code $X_l \supset X_m$ with $l > m$ has also this necklace $nLDCCN$ (Proposition 3.6) and thus, X_l is also not circular. Therefore, all the trinucleotide codes X_m which are not

circular allow to eliminate a huge number of greater trinucleotide codes $X_l \supset X_m$ which are not circular. Storing in memory all these trinucleotide non-circular subcodes X_m and testing their membership of all larger codes X_l is (quite) impossible. Thus, *NA* uses a "branch pruning" method to eliminate efficiently almost all trinucleotide codes sharing a common trinucleotide non-circular subcode.

Furthermore, the algorithm *NA* will be be parallelized in order to benefit from multicore processors.

4.1.2 Trinucleotide code generation

For simplification, the notations and definitions of trinucleotides are presented using the lexicographical order on $\mathcal{A}_4^3$, i.e. $AAA < AAC < \cdots < TTT$. However, the algorithm *NA* is in fact based on conjugate classes of trinucleotides (see Section 4.2) allowing to produce codes without $\{AAA, CCC, GGG, TTT\}$ and conjugate trinucleotides.

Notation 4.1. *The ith trinucleotide T, $T \in \mathcal{A}_4^3$, in the lexicographical order is noted T^i, i.e. $T^1 = AAA$, $T^2 = AAC$, ..., $T^{64} = TTT$.*

Definition 4.2. *In the algorithm NA, a trinucleotide code $X_l = \{T^{i_1}, T^{i_2}, \dots, T^{i_l}\}$ of length l, $l = 1, \dots, 20$, has l distinct trinucleotides T^i with $T^i \in \mathcal{A}_4^3$ and $i_1 < i_2 < \cdots < i_l$.*

A code $X_l = \{T^{i_2}, T^{i_1}, \dots, T^{i_l}\}$ does not verify Definition 4.2.

Example 4.3. *The trinucleotide code $X_3 = \{T^2, T^3, T^{64}\}$ is $\{AAC, AAG, TTT\}$. The trinucleotide code $Y_3 = \{T^3, T^2, T^{64}\} = \{AAG, AAC, TTT\}$ is not considered.*

Furthermore, a trinucleotide code X_l of length l, $l = 1, \dots, 20$, is ordered according to the list of its trinucleotides.

Notation 4.4. *The jth code X_l for a given length l, $l = 1, \dots, 20$, in the lexicographical order is noted X_l^j.*

Example 4.5. *$X_3^{j_1} = \{T^2, T^3, T^{64}\} < X_3^{j_2} = \{T^2, T^4, T^5\}$ for $l = 3$ with $j_1 < j_2$.*

Notation 4.6. *The trinucleotide T^i at the position p, $p = 1, \ldots, l$, in a code X_l of length l is noted T_p^i, i.e. $T_{p_1}^{i_1} < T_{p_2}^{i_2}$ for $1 \leq p_1 < p_2 \leq l$.*

Example 4.7. $X_l = \left\{T_1^{i_1}, \ldots, T_{p_j}^{i_j}, \ldots, T_l^{i_l}\right\}$ *with* $1 \leq j \leq l$.

The first (lower) code of length l, $l = 1, \ldots, 20$, is $X_l^1 = \left\{T_1^1, \ldots, T_l^l\right\}$ and the last (greatest) code of length l is $X_l^{j_{\max}(l)} = \left\{T_1^{64+1-l}, \ldots, T_p^{64+p-l}, \ldots, T_l^{64}\right\}$ with $p = 1, \ldots, l$ and $j_{\max}(l) = \binom{64}{l}$. However, recall that in the effective implementation of the algorithm *NA* (Section 4.2), $j_{\max}(l) = \binom{20}{l} \times 3^l$ (consequence of Proposition 3.5).

Notation 4.8. *A trinucleotide in the configuration T_p^{64+p-l} at position p, $p = 1, \ldots, l$, is called "limit" trinucleotide $T_{\lim}$ of a code X_l.*

Indeed, if a trinucleotide in a position p of a code X_l is lexicographically greater than the limit trinucleotide T_p^{64+p-l} then the code X_l cannot exist as there is not enough greater trinucleotides to complete the positions $p+1, \ldots, l$. In particular, all trinucleotides of a code $X_l^{j_{\max}(l)}$ are limit trinucleotides.

Example 4.9. *The code $X_3 = \{T_1^4, T_2^5, T_3^6\}$ has no limit trinucleotide whereas $Y_3 = \{T_1^4, T_2^{63}, T_3^{64}\}$ has two limit trinucleotides T_2^{63} and T_3^{64}.*

Note that the limit trinucleotides decrease monotonically by 1 from the end of the code.

Notation 4.10. *The pth element of a code X_l, $p = 1, \ldots, l$, is noted $X_l(p)$.*

Example 4.11. *If $X_3 = \{T_1^2, T_2^3, T_3^{64}\}$ then $X_3(2) = T^3$.*

Definition 4.12. *A subcode $X_{m,l}$ of length m of a code X_l of length l with $l = 1, \ldots, 20$ and $0 \leq m < l$, is defined by $X_{m,l} = \{T^{i_1}, T^{i_2}, \ldots, T^{i_m}\}$ where $X_{m,l}(p) = X_l(p)$ with $p = 1, \ldots, m$. A subcode $X_{0,l}$ ($m = 0$) is empty.*

Example 4.13. *The code $X_3 = \{T_1^2, T_2^3, T_3^{64}\}$ has three subcodes: $X_{0,3} = \{\}$, $X_{1,3} = \{T_1^2\}$ and $X_{2,3} = \{T_1^2, T_2^3\}$.*

Trinucleotide codes are sequentially generated by the function *next* $\left(X_l^j\right) = X_l^{j+1}$. Note that *next* $\left(X_l^j\right)$ is undefined for $X_l^{j_{\max}(l)}$. The function *next* $\left(X_l^j\right)$ is algorithmically based on backtracking which generates each X_l^{j+1} by removing some trinucleotides from X_l^j and pushing their immediate successors. Two subfunctions are defined. The subfunction $pop\left(X_l^j\right) = \left(X_{m,l}^{j+1}, T^{last}\right)$ removes $(l-m)$ trinucleotides from a code X_l^j to generate a subcode $X_{m,l}^{j+1}$ with the trinucleotide $T^{last} = X_l^j\,(m+1)$ being the last removed trinucleotide. The subfunction $push\left(X_{m,l}^{j+1}, T^{last}\right) = \left(X_{m+1,l}^{j+1}, T^{last+1}\right)$ pushes the successor of the trinucleotide T^{last}, i.e. T^{last+1}, on a subcode $X_{m,l}^{j+1}$ to generate a subcode $X_{m+1,l}^{j+1}$. If $(m+1) = l$, a complete new code, i.e. X_l^{j+1}, is generated. Note that the *push* function also returns the pushed trinucleotide T^{last+1}.

Definition 4.14. *For a trinucleotide code X_l^j, the subfunction $pop\left(X_l^j\right)$ returning a couple (subcode, last removed trinucleotide) is defined by*

$$pop\left(X_l^j\right) = \begin{cases} \left(X_{l-1,l}^{j+1}, T^{i_l}\right) = \left(\{T_1, \ldots, T_{l-1}\}_{l-1,l}^{j+1}, T^{i_l}\right) \\ \qquad \text{if } X_l^j = \left\{T_1, \ldots, T_{l-1}, T_l^{i_l}\right\} \text{ with } T_l^{i_l} \neq T_{\lim} \\ \left(X_{p-2,l}^{j+1}, T^{i_{p-1}}\right) = \left(\{T_1, \ldots, T_{p-2}\}_{p-2,l}^{j+1}, T^{i_{p-1}}\right) \\ \qquad \text{if } X_l^j = \left\{T_1, \ldots, T_{p-2}, T_{p-1}^{i_{p-1}}, T_p^{(64-l+p)}, \ldots, T_l^{64}\right\} \\ \qquad \text{with } l-p+1 \text{ limit trinucleotides } T_{\lim} \end{cases}$$

Note that the subcode $X_{p-2,l}^{j+1}$ is empty when $p = 2$. Hence, the subfunction $pop\left(X_l^j\right)$ cannot be applied to a code X_l^j with $p < 2$, i.e. with $p = 1$. Indeed, with $p = 1$, there is $l - p + 1 = l$ limit trinucleotides $T_{\lim}$, i.e. $X_l^{j_{\max}(l)}$. Hence, the trinucleotide generation process stops when the code $X_l^j = X_l^{j_{\max}(l)}$.

Definition 4.15. *For a trinucleotide subcode $X_{m,l}^j = \{T_1, \ldots, T_m^{i_m}\}_{m,l}^j$ and a trinucleotide $T^{last} \geq X_{m,l}^j\,(m)$, $i_m \leq last \leq 64 + m - l$, the subfunction $push\left(X_{m,l}^j, T^{last}\right)$ is defined by*

$$push\left(X_{m,l}^j, T^{last}\right) = \begin{cases} \left(X_{m+1,l}^j, T^{last+1}\right) = \left(\left\{T_1, \ldots, T_m, T_{m+1}^{last+1}\right\}_{m+1,l}^j, T^{last+1}\right) \\ \quad \text{if } m < l-1 \\ \left(X_l^j, T^{last+1}\right) = \left(\left\{T_1, \ldots, T_m, T_l^{last+1}\right\}_l^j, T^{last+1}\right) \\ \quad \text{if } m = l-1 \end{cases}$$

The test $last \leq 64 - l + m$ ensures that a complete code X_l^j can be built. It is related to the definition of limit trinucleotides.

Definition 4.16. *Let $first$ be the function defined by $first\,(x, y) = x$ for any pair (x, y).*

Example 4.17. *For a given length $l=3$, let be the code $X_3^1=\{T_1^1,T_2^2,T_3^3\}$. Then, pop $(X_3^1)=(X_{2,3}^2,T^3)=\left(\{T_1^1,T_2^2\}_{2,3}^2,T^3\right)$. Then, push $(X_{2,3}^2,T^3)=\left(\{T_1^1,T_2^2,T_3^{3+1}\},T^{3+1}\right)=(\{T_1^1,T_2^2,T_3^4\},T^4)=(X_3^2,T^4)$ and $first\,(X_3^2,T^4)=X_3^2$.*

Example 4.18. *For a given length $l=3$, let be the code $X_3^{1953}=\{T_1^1,T_2^{63},T_3^{64}\}$ ($\binom{63}{2}=1953$). The last two trinucleotides are limit trinucleotides $T_{\lim}$. Thus, pop $(X_3^{1953})=(X_{0,3}^{1954},T^1)=\left(\{\}_{0,3}^{1954},T^1\right)$. Then, push $(X_{0,3}^{1954},T^1)=\left(\{T_1^2\}_{1,3}^{1954},T^2\right)$. Then, push $\left(\{T_1^2\}_{1,3}^{1954},T^2\right)=\left(\{T_1^2,T_2^3\}_{2,3}^{1954},T^3\right)$ and push $\left(\{T_1^2,T_2^3\}_{2,3}^{1954},T^3\right)=\left(\{T_1^2,T_2^3,T_3^4\}_3^{1954},T^4\right)=(X_3^{1954},T^4)$. Finally, $first\,(X_3^{1954},T^4)=X_3^{1954}$.*

In order to generate a code X_l^{j+1} from a code X_l^j, the subfunction *push* is applied $(l-m)$ times to the result of *pop* $(X_l^j)=(X_{m,l}^{j+1},T^{last})$, i.e. $push^{(l-m)}\left(X_{m,l}^{j+1},T^{last}\right)=\left(X_l^{j+1},T^{i_l}\right)$. This repeated operation is noted $push^*\left(X_{m,l}^{j+1},T^{last}\right)$ according to the Kleene notation.

Definition 4.19. *The function next (X_l^j) generating a new code is defined by*

$$next\left(X_l^j\right)=first\left(push^*\left(pop\left(X_l^j\right)\right)\right)=X_l^{j+1}$$

The repeated application of the function *next* starting with *next* (X_l^1) allows, by construction, to generate uniquely all trinucleotide codes of length l in the lexicographical order.

4.1.3 The necklace test

The necklace test is presented here under the assumption that the trinucleotide codes contain neither $\{AAA,CCC,GGG,TTT\}$ nor conjugate trinucleotides, i.e. they have no necklace $2LDCCN$ (Proposition 3.5). Let $l\in\mathcal{A}_4$ be a letter (nucleotide) and $d\in\mathcal{A}_4^2$, a diletter (dinucleotide).

Notation 4.20. *A trinucleotide T^i in the lexicographical order is noted $T^i=l^id^i$. Similarly, a trinucleotide T_p at position p in a code X_l^j is noted $T_p=l_pd_p$.*

Notation 4.21. *Two trinucleotides* $T^i = l^i d^i$ *and* $T^j = l^j d^j$ *can be combined into two new trinucleotides* $T^{i,j} = d^i l^j$ *and* $T^{j,i} = d^j l^i$. *Two trinucleotides* $T_p = l_p d_p$ *and* $T_q = l_q d_q$ *at the positions* p *and* q *in a code* X_l^j *can be combined into two new trinucleotides* $T_{p,q} = d_p l_q$ *and* $T_{q,p} = d_q l_p$.

Definition 4.22. *The generated trinucleotide matrix* $\mathcal{M}\left(X_l^j\right) = \mathcal{M}_l^j$ *of a code* $X_l^j = \{l_1 d_1, \ldots, l_p d_p, \ldots, l_l d_l\}$ *is defined by the trinucleotides* $T_{p,q}$ *and* $T_{q,p}$ *at row* p *and column* q*, and row* q *and column* p*, respectively,*

$$\mathcal{M}\left(X_l^j\right) = \mathcal{M}_l^j = \begin{pmatrix} \varnothing & d_1 l_2 & \cdots & d_1 l_p & \cdots & \cdots & d_1 l_l \\ d_2 l_1 & \varnothing & & \vdots & & & \vdots \\ \vdots & & \ddots & \vdots & & & \vdots \\ d_p l_1 & \cdots & \cdots & \varnothing & \cdots & \cdots & d_p l_l \\ \vdots & & & \vdots & \ddots & & \vdots \\ \vdots & & & \vdots & & \varnothing & d_{l-1} l_l \\ d_l l_1 & \cdots & \cdots & d_l l_p & \cdots & d_l l_{l-1} & \varnothing \end{pmatrix}$$

The main diagonal is empty: the trinucleotides $d_p l_p$, $p = 1, \ldots, l$, *are not generated as the necklace* $2LDCCN$ *is not tested (Section 4.1.1).*

Example 4.23. *The generated trinucleotide matrix* $\mathcal{M}\left(X_l^j\right) = \mathcal{M}_3^j$ *of the code* $X_3^j = \{l_1 d_1, l_2 d_2, l_3 d_3\}$ *of length* $l = 3$, *is*

$$\begin{array}{c c} & \begin{array}{ccc} l_1 d_1 & l_2 d_2 & l_3 d_3 \end{array} \\ \begin{array}{c} l_1 d_1 \\ l_2 d_2 \\ l_3 d_3 \end{array} & \begin{pmatrix} \varnothing & d_1 l_2 & d_1 l_3 \\ d_2 l_1 & \varnothing & d_2 l_3 \\ d_3 l_1 & d_3 l_2 & \varnothing \end{pmatrix} \end{array}$$

Following Definition 3.3 (the indices of letters and diletters are not associated to a position in a code), a trinucleotide code $X_l^j = \{l_1 d_1, l_2 d_2, l_3 d_3, \ldots, l_l d_l\}$ has a necklace $(n+1)\, LDCCN$ if an ordered sequence $\mathcal{S} = l_{p_1}, d_{p_1}, \ldots, l_{p_n}, d_{p_n}, l_{p_1}$ of letters and diletters have trinucleotides $T \in X_l^j$.

Remark 4.24. *The sequence* $\mathcal{S} = l_1, d_1, l_3, d_3, l_1$ *can potentially lead to a necklace whereas the sequence* l_1, d_1, l_2, d_3, l_1 *cannot and is never constructed by the algorithm NA.*

Definition 4.25. *The sequence* $\mathcal{S}' = d_{p_1}, l_{p_2}, \ldots, d_{p_n}, l_{p_1}$ *is deduced from the sequence* $S = l_{p_1}, d_{p_1}, l_{p_2}, \ldots, l_{p_n}, d_{p_n}, l_{p_1}$ *by removing the first letter* l_{p_1}. *The* *sequence $\mathcal{S}''$ of n trinucleotides (n being related to the $(n+1)\,LDCCN$) is deduced from the sequence $\mathcal{S}'$ where each d_{p_i}, l_{p_j} is replaced by the trinucleotide $T_{i,j}$.*

Example 4.26. *The sequence $\mathcal{S} = l_1, d_1, l_3, d_3, l_1$ is associated to the sequences $\mathcal{S}' = d_1, l_3, d_3, l_1$ and $\mathcal{S}'' = T_{1,3}, T_{3,1}$ of two trinucleotides.*

The generated trinucleotide matrix $\mathcal{M}\left(X_l^j\right)$ contains all relevant combinations of trinucleotides of X_l^j. If the n trinucleotides of the sequence $\mathcal{S}''$ belong to X_l^j, then the code X_l^j has a necklace $(n+1)\,LDCCN$ and thus is not circular.

Example 4.27. *If $T_{1,3} = \mathcal{M}\left(X_l^j\right)_{1,3}$ and $T_{3,1} = \mathcal{M}\left(X_l^j\right)_{3,1}$ belong to X_l^j, then X_l^j has a necklace $(2+1)\,LDCCN$ and thus is not circular.*

As a conclusion, the search for a necklace $(n+1)\,LDCCN$ in a code X_l^j is equivalent to search a sequence
$\mathcal{S}'' = T_{p_1,p_2}, T_{p_2,p_3}, \ldots, T_{p_i,p_{i+1}}, T_{p_{i+1},p_{i+2}}, \ldots, T_{p_{n-1},p_n}, T_{p_n,p_{n+1}}$ with
$T_{p_i,p_{i+1}} \in \mathcal{M}\left(X_l^j\right)$ and $T_{p_i,p_{i+1}} \in X_l^j$ for all $1 \le i \le n$ and $p_{n+1} = p_1$.

Notation 4.28. *A trinucleotide chain $\mathcal{C}\left(X_l^j\right)$ of a code X_l^j is a sequence $\mathcal{M}\left(X_l^j\right)_{p_1,p_2}, \ldots, \mathcal{M}\left(X_l^j\right)_{p_i,p_{i+1}}, \mathcal{M}\left(X_l^j\right)_{p_{i+1},p_{i+2}}, \ldots, \mathcal{M}\left(X_l^j\right)_{p_n,p_{n+1}}$ with distinct $1 \le p_i \le l$ of n trinucleotides such that $\mathcal{M}\left(X_l^j\right)_{p_i,p_{i+1}} \in X_l^j$.*

Definition 4.29. *For a code X_l^j, the trinucleotide chain $\mathcal{C}'\left(X_l^j\right) = \mathcal{C}\left(X_l^j\right) \cdot T_{p_{n+1},p_{n+2}}$ is composed of the trinucleotide $T_{p_{n+1},p_{n+2}} \in \mathcal{M}\left(X_l^j\right)$ added at the end of $\mathcal{C}\left(X_l^j\right) = \mathcal{M}\left(X_l^j\right)_{p_1,p_2}, \ldots, \mathcal{M}\left(X_l^j\right)_{p_n,p_{n+1}}$ with distinct $1 \le p_i \le l$.*

Remark 4.30. *The operation $\cdot$ is used between a chain, where trinucleotides are separated by commas, and a trinucleotide which is added at the end of the chain.*

Example 4.31. *For a code $X_4^j = \{l_1 d_1, l_2 d_2, l_3 d_3, l_4 d_4\}$ and a trinucleotide chain $\mathcal{C}\left(X_l^j\right) = T_{1,2}, T_{2,3}$, $\mathcal{C}'\left(X_l^j\right) = \mathcal{C}\left(X_l^j\right) \cdot T_{3,4} = T_{1,2}, T_{2,3}, T_{3,4}$ with $T_{i,j} = \mathcal{M}\left(X_l^j\right)_{i,j}$.*

Definition 4.32. *For a code X_l^j, a trinucleotide chain $\mathcal{C}\left(X_l^j\right)$ of length n is closed when its first trinucleotide is $\mathcal{M}\left(X_l^j\right)_{p_1,p_2}$ and its last trinucleotide is $\mathcal{M}\left(X_l^j\right)_{p_n,p_1}$, forming a sequence S'', i.e. with a necklace $(n+1)\,LDCCN$.*

Definition 4.33. *For a code X_l^j and a trinucleotide chain $\mathcal{C}\left(X_l^j\right) = \mathcal{M}\left(X_l^j\right)_{p_1,p_2}, \ldots, \mathcal{M}\left(X_l^j\right)_{p_n,p_{n+1}}$, the trinucleotide $\mathcal{M}\left(X_l^j\right)_{p_1,p_2}$ is the seed of the chain.*

In our case, only closed chains (i.e. sequences S'') of length 2, 3 and 4 are searched (Proposition 3.4). Precisely, only one closed chain has to be found to prove that a code is not circular. Thus, the boolean function $chain\left(Start, Current, \mathcal{C}, X_l^j\right)$ searchs for a closed chain $\mathcal{C}$ in a trinucleotide code X_l^j, returning $True$ as soon as a closed chain is found ($Start$ and $Current$ are indices defined below).
The function *chain* uses the function $row\left(Current, X_l^j\right)$ defined as follows.

Definition 4.34. *For a code X_l^j, $row\left(Current, X_l^j\right)$, $1 \leq Current \leq l$, gives the set of trinucleotides from the Current row of $\mathcal{M}\left(X_l^j\right)$ belonging to X_l^j*

$$row\left(Current, X_l^j\right) = \left\{ \mathcal{M}\left(X_l^j\right)_{Current,Col} \middle| \begin{array}{c} \mathcal{M}\left(X_l^j\right)_{Current,Col} \in X_l^j, 1 \leq Col \leq l \\ with\ Col \neq Current \end{array} \right\}$$

Definition 4.35. *For a trinucleotide code X_l^j, the boolean function $chain\left(Start, Current, \mathcal{C}, X_l^j\right)$, $1 \leq |\mathcal{C}| \leq 3$, $\mathcal{C} = \mathcal{M}\left(X_l^j\right)_{Start,p_2}, \ldots,$ $\mathcal{M}\left(X_l^j\right)_{p_n,Current}$, $1 \leq Start \leq l$, $1 \leq Current \leq l$ and $Start \neq Current$, is defined by*

$$chain\left(Start, Current, \mathcal{C}, X_l^j\right) = \begin{cases} True\ if\ |\mathcal{C}| < 3\ and\ isClosed = True \\ \displaystyle\bigvee_{T_{Current,Col} \in row\left(Current, X_l^j\right)} chain\left(Start, Col, \mathcal{C} \cdot T_{Current,Col}, X_l^j\right) \\ \quad if\ |\mathcal{C}| < 3\ and\ isClosed = False\ and\ \left|row\left(Current, X_l^j\right)\right| > 0 \\ False\ if\ |\mathcal{C}| < 3\ and\ isClosed = False\ and\ \left|row\left(Current, X_l^j\right)\right| = 0 \\ True\ if\ |\mathcal{C}| = 3\ and\ isClosed = True \\ False\ otherwise \end{cases}$$

with the boolean predicate $isClosed = \mathcal{M}\left(X_l^j\right)_{Current,Start} \in X_l^j$.

The function *chain* is a recursive boolean function that grows an initial chain $\mathcal{C}$ from a seed belonging to X_l^j. This seed is mandatory and, according to the preconditions, the initial call to *chain* must have the pattern

$chain\left(Start, Current, \mathcal{M}\big(X_l^j\big)_{Start,Current}, X_l^j\right)$. The seed matches with the predicate $isClosed = \mathcal{M}\big(X_l^j\big)_{Current,Start} \in X_l^j$ which is currently testing if a closed chain exists. At each recursive call, *chain* first tries to grow the chain $\mathcal{C}$ by closing it with $\mathcal{M}\big(X_l^j\big)_{Current,Start}$. If *isClosed* succeeds then it exits a closed chain $\mathcal{C} \cdot \mathcal{M}\big(X_l^j\big)_{Current,Start}$ of length $|\mathcal{C}| + 1 = n$ trinucleotides that belongs to X_l^j. Thus, X_l^j has at least one necklace $(n+1)\,LDCCN$. If *isClosed* fails then there are two cases. If $|\mathcal{C}| = 3$ trinucleotides then $False$ is returned as a closing chain of length $|\mathcal{C}| + 1 = 4$ trinucleotides does not exist for $\mathcal{C}$ and, by definition, there is no need to search for longer closed chains as the maximum length of $|\mathcal{C}|$ is 4 trinucleotides. On the other hand, if $|\mathcal{C}| < 3$ trinucleotides then longer chains starting with $\mathcal{C}$ must be built and tested.

There is a link between $Current$ and $\mathcal{C}$ parameters of the function *chain*. Indeed, $\mathcal{C}$ always has the pattern $\mathcal{C} = \mathcal{M}\big(X_l^j\big)_{Start,p_1}, \ldots, \mathcal{M}\big(X_l^j\big)_{p_n,Current}$, i.e. it ends with a trinucleotide from the $Current$ column of $\mathcal{M}\left(X_l^j\right)$. Thus, $\mathcal{C}$ can grow by concatenating a trinucleotide from the $Current$ row of $\mathcal{M}\left(X_l^j\right)$. Note that this remark also applies to the predicate *isClosed*. Thus, the function $row\left(Current, X_l^j\right)$ is defined in order to return the set of trinucleotides $T_{Current,Col}$ belonging both to the current row and the code X_l^j. The column Col of the trinucleotide $T_{Current,Col}$, the last trinucleotide of the chain $\mathcal{C} \cdot T_{Current,Col}$, is passed to the recursive function
$chain\left(Start, Col, \mathcal{C} \cdot T_{Current,Col}, X_l^j\right)$. As there is $\left|row\left(Current, X_l^j\right)\right|$ recursive function calls, i.e. a call per item of the set $row\left(Current, X_l^j\right)$, a logical boolean operator OR is used to gather all results in one boolean. If a recursive function call returns $True$ then there is a necklace and the result of the current *chain* is $True$. If all recursive function calls fail or $row\left(Current, X_l^j\right)$ is empty, then there is no necklace and the result of the current *chain* function call is $False$.

Example 4.36. *Based on the matrix of Example 4.23 with the code* $X_3^j = \{l_1d_1, l_2d_2, l_3d_3\}$ *of length* $l = 3$ *trinucleotides and the initial call chain* $\left(1, 2, \mathcal{C} = d_1l_2, X_3^j\right)$. *As* $|\mathcal{C}| < 3$, *isClosed is tested. If isClosed is true then* X_3^j *has a necklace* $(2+1)\,LDCCN$ *associated with the trinucleotide sequence* d_1l_2, d_2l_1 *from the matrix* $\mathcal{M}\left(X_l^j\right)$. *If isClosed is False then the row 2 is searched for a trinucleotide* $T_{2,Col} \in X_3^j$. *The only possibility here is* $T_{2,Col} = d_2l_3$ *as isClosed is False then* $d_2l_1 \notin X_3^j$. *If* $d_2l_3 \notin X_3^j$, $False$ *is returned and the code* X_3^j *does not have a closed chain starting with the seed*

d_1l_2. If $d_2l_3 \in X_3^j$, the call chain $\left(1, 3, \mathcal{C} = (d_1l_2, d_2l_3), X_3^j\right)$ is performed. The trinucleotide d_3l_1 is tested by isClosed. If isClosed is true then X_3^j has a necklace $(3+1)\,LDCCN$ associated with the trinucleotide sequence d_1l_2, d_2l_3, d_3l_1 from the matrix $\mathcal{M}\left(X_l^j\right)$, else the call chain $\left(1, 2, \mathcal{C} = (d_1l_2, d_2l_3, d_3l_2), X_3^j\right)$ is performed.

Two important points should be adressed. Firstly, the chain d_1l_2, d_2l_3, d_3l_2 has a length of 3 trinucleotides, thus it is the last recursive call. Secondly, if the factor $d_2l_3d_3l_2$ of the chain d_1l_2, d_2l_3, d_3l_2 is a necklace $3LDCCN$ then it is not identified with the last call which only tries to close the chain with d_2l_1 without success. Further calls of the function chain based on different seeds are necessary to identify the necklace, e.g. chain $\left(2, 3, \mathcal{C} = d_2l_3, X_3^j\right)$. Figure 1 illustrates the function chain with the dotted arrows for the search of the necklace $3LDCCN$ associated with the trinucleotide sequence d_1l_2, d_2l_1 from the matrix $\mathcal{M}\left(X_l^j\right)$ and the plain arrows for the search of the necklace $4LDCCN$ associated with d_1l_2, d_2l_3, d_3l_1.

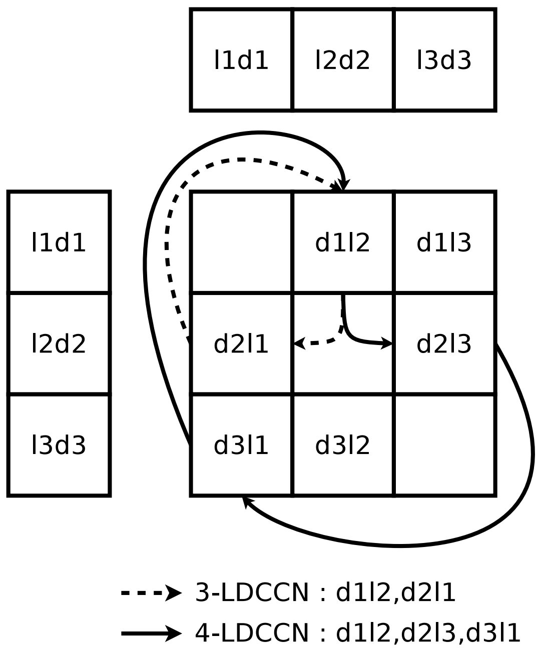

Figure 1: A search example of necklaces $(2+1)\,LDCCN$ and $(3+1)\,LDCCN$ in the generated trinucleotide matrix $\mathcal{M}\left(X_l^j\right)$ of Example 4.23.

A function *test* $\left(X_l^j\right)$ browses the matrix $\mathcal{M}\left(X_l^j\right)$ using each trinucleotide as a seed for a call to the function *chain*. However, unnecessary tests can be avoided. Firstly, the seed must belong to X_l^j. Secondly, the test is symmetric with respect to the main diagonal of $\mathcal{M}\left(X_l^j\right)$.

Example 4.37. *With $X_3^j = \{l_1d_1, l_2d_2, l_3d_3\}$, the necklace $3LDCCN$ associated with the trinucleotide sequence d_2l_3, d_3l_2 from the matrix $\mathcal{M}\left(X_l^j\right)$ can be found either with the seed d_2l_3 yielding to the necklace associated with d_2l_3, d_3l_2 or with the seed d_3l_2 yielding to the necklace associated with d_3l_2, d_2l_3.*

More generally, thanks to this matrix symmetry, only trinucleotides above (or under) the main diagonal of $\mathcal{M}\left(X_l^j\right)$ are needed as seed to identify necklaces. The boolean function $test\left(X_l^j\right)$ returns $True$ if the code X_l^j has a necklace $(n+1)\,LDCCN$, $n \in \{2,3,4\}$, i.e. is not circular, and $False$ otherwise. It only uses trinucleotides above the main diagonal of $\mathcal{M}\left(X_l^j\right)$ as seed of chains $\mathcal{C}$ and is defined as follows.

Definition 4.38. *For a trinucleotide code X_l^j, the boolean function test $\left(X_l^j\right)$ is*

$$test\left(X_l^j\right) =$$

$$\begin{cases} \displaystyle\bigvee_{\substack{1 \le Start < l \\ Start < Current \le l \\ \mathcal{M}\left(X_l^j\right)_{Start,Current} \in X_l^j}} chain\left(Start, Current, \mathcal{M}\left(X_l^j\right)_{Start,Current}, X_l^j\right) & \text{if } l > 1 \\ False \text{ if } l = 1 \end{cases}$$

Note that, as the trinucleotides $\{AAA, CCC, GGG, TTT\}$ are not considered, returning $False$ for $l = 1$ is correct.

4.1.4 Branch pruning

The trinucleotide code generation (Section 4.1.2) can be associated to a forest $\mathcal{F}_l$ of trinucleotide codes of a given length l. This forest $\mathcal{F}_l$ is a disjoint union of trinucleotide trees $\mathcal{T}_l$. Its structure allows the branch pruning method which is now described. Figure 2 gives an illustration of such a forest.

A trinucleotide code X_l^j represents a complete branch of a tree $\mathcal{T}_l^t$ in $\mathcal{F}_l$, $1 \le t \le (64+1-l)$, from the root $X_l^j(1) = T_1^t$ to the leaf $X_l^j(l)$ with internal nodes $X_l^j(m)$ at depth $1 \le m < l$. In particular, if $l = 1$ then the tree has only one trinucleotide which is the root of the tree itself, i.e. $X_1^j(1) = T_1^t$. The trees $\mathcal{T}_l^t$ are ordered by their root trinucleotides T_1^t, from left to right.

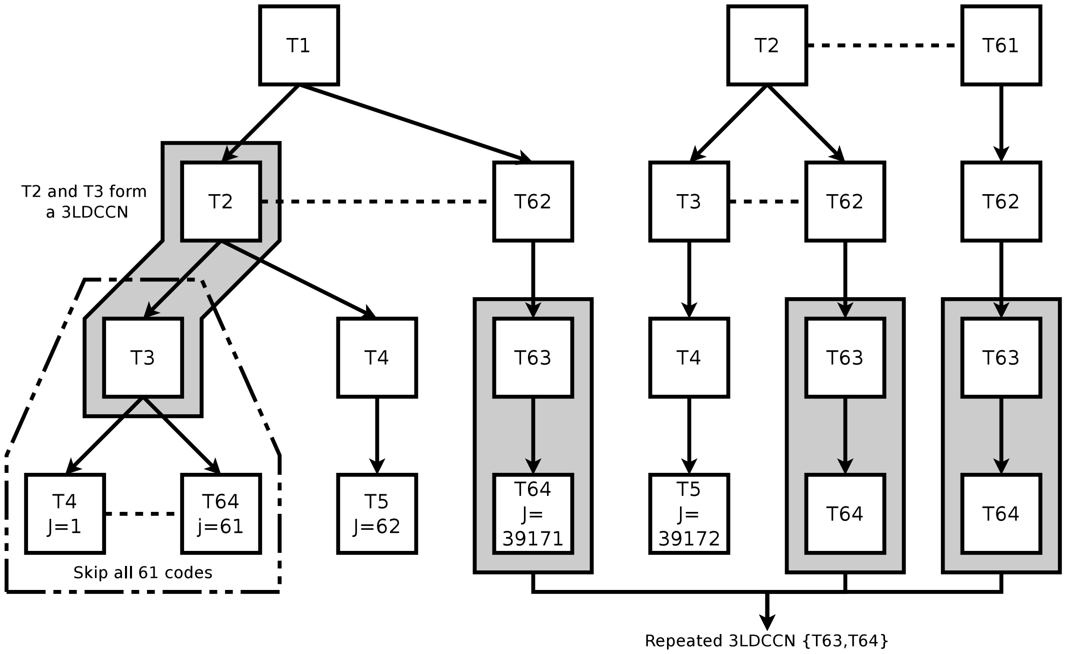

Figure 2: Example of a forest for the 64 trinucleotides $T \in \mathcal{A}_4^3$ of length $l = 4$. Dotted line represent non-contiguous branches. A grey background represent a necklace. The dotted trapezoid shows branch pruning caused by a necklace detected on an incomplete branch.

Thus, a forest $\mathcal{F}_l$ has $(64 + 1 - l)$ distinct trees, the last tree $\mathcal{T}_l^{64+1-l}$ being the root $X_l^{j_{\max}}(1) = T_1^{64+1-l}$ which is, by definition, the limit trinucleotide $T_{\lim} = T_p^{64+p-l}$ at position $p = 1$. The tree $\mathcal{T}_l^t$ has $|\mathcal{T}_l^t| = \binom{64-t}{l-1}$ distinct codes of length l as the first trinucleotide, i.e. T_1^t, is fixed by the tree itself. In a given tree, codes are also ordered from left to right.

A nonempty subcode $X_{m,l}^j \in \mathcal{T}_l^t$ in $\mathcal{F}_l$ with $1 \leq m < l$ is an incomplete branch from the root $X_{m,l}^j(1) = X_l^j(1) = T_1^t$ to an internal node $X_{m,l}^j(m) = X_l^j(m)$ at depth m. A subcode is usually shared by several complete distinct codes from the same tree $\mathcal{T}_l^t$ in $\mathcal{F}_l$, i.e. for two distinct codes $X_l^{j_1}$ and $X_l^{j_2}$, $j_1 \neq j_2$ in $\mathcal{T}_l^t$, $X_{m,l}^{j_1} = X_{m,l}^{j_2}$ are the same incomplete branch from the root $X_{m,l}^{j_1}(1) = X_{m,l}^{j_2}(1) = \{T_1^t\}$ to the internal node $X_{m,l}^{j_1}(m) = X_{m,l}^{j_2}(m)$ for some given m with $1 \leq m < l$. In particular, they always share the same root of $\mathcal{T}_l^t$, i.e. $X_{1,l}^{j_1} = X_{1,l}^{j_2} = X_l^{j_1}(1) = X_l^{j_2}(1) = T_1^t$. If $m = l - 1$, the two codes only differ by their last trinucleotides. Thus, an incomplete branch (subcode) $X_{m,l}$ in a tree $\mathcal{T}_l^t$ participates to a set of contiguous complete branches (codes) noted

$(X_{m,l})^*$. The cardinality of $(X_{m,l})^*$ is $|X_{m,l}| = \binom{64-i_m}{l-m}$ with $X_{m,l}(m) = T_m^{i_m}$ being the i_mth trinucleotide in the alphabet. So, several codes $X_l^I \in (X_{m,l})^*$, $I = \{i, i+1, i+2, \ldots, i + |X_{m,l}| - 1\}$, share the same subcode $X_{m,l}^i$. As these codes X_l^I are contiguous, the subcode number is the number of the lowest complete code sharing it, i.e. $X_{m,l}^i = X_{m,l}^{\min(I)}$. All complete branches from $(X_{m,l})^*$ can be skipped in the trinucleotide generation process by applying the function *next* to the subcode $X_{m,l}$.

Example 4.39. *In a forest for the* 64 *trinucleotides* $T \in \mathcal{A}_4^3$ *of length* $l = 4$ *(Figure 2), let be three trinucleotide codes* $X_4^1 = \{T^1, T^2, T^3, T^4\}$ *in the tree* $\mathcal{T}_4^1$, $X_4^{62} = \{T^1, T^2, T^4, T^5\}$ *in the tree* $\mathcal{T}_4^1$ *and* $X_4^{39712} = \{T^2, T^3, T^4, T^5\}$ *in the tree* $\mathcal{T}_4^2$. *The first code in* $\mathcal{T}_4^2$ *has the number* $|\mathcal{T}_l^t|+1 = \binom{64-1}{4-1}+1 = \binom{63}{3}+1 = 39172$ *(*$t = 1$ *and* $l = 4$*), i.e. the code* X_4^{39712}. *The two codes* X_4^1 *and* X_4^{62} *share two nonempty subcodes, i.e.* $X_{1,4}^1 = X_{1,4}^{62} = \{T^1\}$ *and* $X_{2,4}^1 = X_{2,4}^{62} = \{T^1, T^2\}$ *which are in the same tree* $\mathcal{T}_4^1$. *The incomplete branch* $X_{3,4} = \{T^1, T^2, T^3\}_{3,4}$ *in the tree* $\mathcal{T}_4^1$ *yields to* $|X_{m,l}| = \binom{64-i_m}{l-m} = \binom{64-3}{4-3} = 61$ *complete branches (*$l = 4$, $m = 3$ *and* $i_m = 3$*), i.e.* $\left(\{T^1, T^2, T^3\}_{3,4}\right)^*$ *has* 61 *codes. Hence, the first complete branch of the next incomplete branch* $\{T^1, T^2, T^4\}_{3,4}$ *has the number* 62. *As a consequence, if* $\{T^1, T^2, T^3\}_{3,4}$ *has a necklace then* 61 *codes can be skipped in one step. The code* X_4^{39712} *in the tree* $\mathcal{T}_4^2$ *shares no nonempty subcodes with the codes* $X_4^1, \ldots, X_4^{62}$ *in the tree* $\mathcal{T}_4^1$ *as their trees are distinct. However, due to the trinucleotide generation scheme, not all subcodes with necklaces allow to prune branches definitively. For example, if the trinucleotides* T^{63} *and* T^{64} *form a necklace* $3LDCCN$, *all codes containing* T^{63} *and* T^{64} *will be however generated. Note that the number* $|\mathcal{T}_l^t|$ *of codes per tree decreases, i.e.* $\left|\mathcal{T}_l^{t_1}\right| > \left|\mathcal{T}_l^{t_2}\right|$ *for* $t_1 < t_2$. *The last tree* $\mathcal{T}_4^{61}$ *is composed of only one code, i.e.* $\{T^{61}, T^{62}, T^{63}, T^{64}\}$.

The principle of the algorithm *NA* consists in generating trinucleotide codes X_l^j for a given length l. Then, for each code X_l^j, the associated generated trinucleotide matrix $\mathcal{M}\left(X_l^j\right)$ is built and the $LDCCN$ necklace test is performed. However, several subcodes $X_{m,l}^j$ are previously generated. According to Proposition 3.6, if a subcode $X_{m,l}^j$ has a necklace then the code X_l^j has also a necklace. Thus, performing the necklace test on subcodes allows to avoid the complete generation of the non circular code X_l^j. Furthermore, $\left|X_{m,l}^j\right|$ contiguous codes share the same subcode $X_{m,l}^j$. Hence, all $\left|X_{m,l}^j\right|$ contiguous codes can be skipped by pruning the incomplete branch $X_{m,l}^{j}$ and carrying on the calculation process on the next incomplete branch $X_{m,l}^{j_{next}}$. If $X_{m,l}^{j_{next}}$ does not exist then the computation stops. Thus, several new subfunctions must be defined for subcodes.

Definition 4.40. *The function* $next'\left(X_{m,l}^{j}\right)$ *generating a new code* X_{l}^{j} *is defined by*

$$next'\left(X_{m,l}^{j}\right) = push'^{*}\left(pop'\left(X_{m,l}^{j}\right)\right) = \left(X_{l}^{j_{next}}, T^{last}\right)$$

Definition 4.41. *For a trinucleotide subcode* $X_{m,l}^{j} = \{T_1, \ldots, T_m^{i_m}\}_{m,l}^{j}$*, the subfunction* $pop'\left(X_{m,l}^{j}\right)$ *returning a couple (subcode, last removed trinucleotide) is defined by*

$$pop'\left(X_{m,l}^{j}\right) = \begin{cases} \left(X_{m-1,l}^{j+1}, T^{i_m}\right) = \left(\{T_1, \ldots, T_{m-1}\}_{m-1,l}^{j+1}, T^{i_m}\right) \\ \qquad \textit{if } X_{m,l}^{j} = \{T_1, \ldots, T_{m-1}, T_m^{i_m}\} \ \textit{ and } T_m^{i_m} \neq T_{\lim} \\ \left(X_{p-2,l}^{j+1}, T^{i_{p-1}}\right) = \left(\{T_1, \ldots, T_{p-2}\}_{p-2,l}^{j+1}, T^{i_{p-1}}\right) \\ \qquad \textit{if } X_{m,l}^{j} = \left\{T_1, \ldots, T_{p-2}, T_{p-1}^{i_{p-1}}, T_p^{(64-l+p)}, \ldots, T_m^{(64-l+m)}\right\} \\ \qquad \textit{with } m-p+1 \textit{ limit trinucleotides } T_{\lim} \end{cases}$$

The definition of the function $pop'\left(X_{m,l}^{j}\right)$ is closed to the definition of the function $pop\left(X_{l}^{j}\right)$. However, although the subcode length is equal to m, the limit trinucleotide is still defined on the code of length l. As with $pop\left(X_{l}^{j}\right)$, the function $pop'\left(X_{m,l}^{j}\right)$ is undefined when the subcode $X_{m,l}^{j}$ contains only limit trinucleotides, stopping the process. The new subfunction $push'\left(X_{m,l}^{j}, T^{last}\right)$ tests if the subcode $X_{m,l}^{j}$ has a necklace before pushing the trinucleotide T^{last} on $X_{m,l}^{j}$. If the subcode $X_{m,l}^{j}$ has a necklace then the next subcode without a necklace must be computed. It is not necessary to redefine the function *test* for subcodes. Indeed, a subcode $X_{m,l}^{j}$ can arbitrarily be considered as a code Y_m of length m, i.e. $X_{m,l}^{j} = \left\{T_1^{i_1}, \ldots, T_m^{i_m}\right\} = Y_m$ and tested with $test\left(Y_m\right)$. Hence, if a subcode $X_{m,l}^{j}$ has a necklace then no shorter subcode $X_{n,l}^{j}$ with $n < m$ has a necklace.

Definition 4.42. *For a trinucleotide subcode* $X_{m,l}^{j} = \{T_1, \ldots, T_m^{i_m}\}_{m,l}^{j}$ *and a trinucleotide* $T^{last} \geq X_{m,l}^{j}(m)$*,* $i_m \leq last \leq 64 - l + m$*, the subfunction* $push'\left(X_{m,l}^{j}, T^{last}\right)$ *is defined by*

$$push'\left(X_{m,l}^{j}, T^{last}\right) = \begin{cases} push\left(X_{m,l}^{j}, T^{last}\right) \ \textit{ if } X_{m,l}^{j} \textit{ is circular (no necklace)} \\ next'\left(X_{m,l}^{j}\right) \ \textit{ otherwise} \end{cases}$$

Definition 4.43. *The function* $next''\left(X_l^j\right)$ *generating a new complete code* X_l^j *is defined by*

$$next''\left(X_l^j\right) = first\left(push'^*\left(pop\left(X_l^j\right)\right)\right) = X_l^{j_{next}}$$

Note that the first call of the function *pop* is the same as the one in Definition 4.14 because the function $next''$ applies on a complete code. The definition of the value of j_{next} is more technical than the one in Definition 4.19 with the function *next* as now several codes can be skipped. A first way is to count the number $\left|X_{m,l}^j\right|$ of skipped codes each time a subcode $X_{m,l}^j$ has a necklace in the function $push'$. A second way is to compute j_{next} from the trinucleotides composing the code itself.

Definition 4.44. *For a code* $X_l^j = \left\{T_1^{i_1}, \ldots, T_l^{i_l}\right\}_l^j$*, its number* j *is defined by*

$$j = number\left(X_l\right) = 1 + \sum_{p=1}^{l} \sum_{\delta=1}^{\Delta} \binom{64 - i_{p-1} - \delta}{l - p} \tag{1}$$

with $\Delta = i_p - i_{p-1} - 1$*,* $X_l\left(p\right) = T_p^{i_p}$*,* $X_l\left(p-1\right) = T_{p-1}^{i_{p-1}}$ *and* $X_l\left(0\right) = T_0^0$*, i.e.* $i_0 = 0$*.*

Intuitively, in the tree representation where trees and codes are ordered from left to right, the number j of the code X_l^j in a given tree is the number of all previous codes plus one. In Example 4.39, the first code $\{T^1, T^2, T^3, T^4\}$ has no previous code/left branch. Its number is $1 + 0 = 1$. For the first code of the second tree $\{T^2, T^3, T^4, T^5\}$, the previous codes are all codes from the first tree. Its number is $1 + \binom{63}{3} = 39172$. For the first code of the third tree $\{T^3, T^4, T^5, T^6\}$, the previous codes are all codes from the first and the second trees. Its number is $1 + \binom{63}{3} + \binom{62}{3} = 1 + 39171 + 37820 = 76992$. The number of previous codes in the current tree is obtained with the number $\left|X_{m,l}\right|$ of complete branches $X_{m,l}$ that shared the incomplete branch $X_{m,l}^j$, i.e. $\left|X_{m,l}\right| = \binom{64-i_m}{l-m}$ with i_m being the ordering number of the last trinucleotide of $X_{m,l}^j$ ($X_{m,l}^j\left(m\right) = T_m^{i_m}$). A virtual trinucleotide T^0 which is the root of a new tree $\mathcal{T}_{l+1}^0$ built from the forest $\mathcal{F}_l$ allows to apply the formula to tree roots (Figure 3).

The double sum in Formula 1 stands for two dimensions: the current trinucleotide depth in the tree, i.e. $\sum_{p=1}^{l}$, and the gap width to the leftmost possible position at that depth, i.e. $\sum_{\delta=1}^{\Delta}$with $\Delta = i_p - i_{p-1} - 1$ (dotted arrows labelled

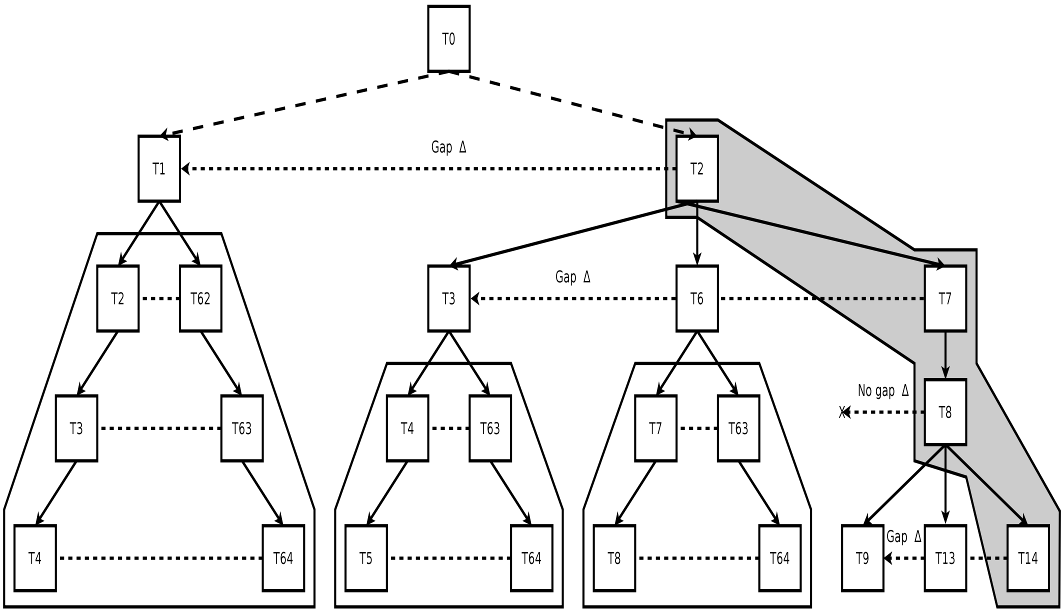

Figure 3: Representation of gaps Δ for calculating the number j of the code $X_4^j = \{T^2, T^7, T^8, T^{14}\}$ (grey background). The forest $\mathcal{F}_l$ is anchored in a new tree $\mathcal{T}_{l+1}^0$ with a virtual trinucleotide T^0. Dotted arrows represent gaps, plain arrows represent real links, dashed arrows represent virtual links and trapezoids represent $(X_{m,l})^*$ sets. The number of the code $X_4^j = \{T^2, T^7, T^8, T^{14}\}$ is determined in Example 4.45.

"Gap Δ" in Figure 3). The gap formula $\Delta = i_p - i_{p-1} - 1$ is determined according to the parent node of the current node: by construction, the leftmost son of an internal node $T_p^{i_p}$ in $\mathcal{T}_{l+1}^0$ is $T_{p+1}^{i_p+1}$, i.e. $i_{p+1} = i_p + 1$. As a consequence, Δ is never negative. So, for a given depth p, the gap represents the number Δ of distinct previous incomplete branches $X_{p,l} = \left\{T_1^{i_1}, \ldots, T_{p-1}^{i_{p-1}}, T_p^{i_{p-1}+\delta}\right\}$ with $1 \leq \delta \leq \Delta$ of $X_{p,l} = \left\{T_1^{i_1}, \ldots, T_{p-1}^{i_{p-1}}, T_p^{i_p}\right\}$. For each incomplete branch, the number $|X_{p,l}| = \binom{64-i_{p-1}-\delta}{l-p}$ must be computed leading to the term $\sum_{\delta=1}^{\Delta} \binom{64-i_{p-1}-\delta}{l-p}$.

Example 4.45. *For the code $X_4^j = \{T^2, T^7, T^8, T^{14}\}$ (in grey in Figure 3), $j = 46141$, i.e. $X_4^{46141} = \{T^2, T^7, T^8, T^{14}\}$. Indeed, by computing Formula 1, we have the following partial results.*
For $p = 1$, $\Delta = i_1 - i_0 - 1 = 2 - 0 - 1 = 1$ and for $\delta = 1$, $\binom{64-0-1}{4-1} = \binom{63}{3} = 39171$.
For $p = 2$, $\Delta = i_2 - i_1 - 1 = 7 - 2 - 1 = 4$ and for $\delta = 1$, $\binom{64-2-1}{4-2} = \binom{61}{2} = 1830$, for $\delta = 2$, $\binom{60}{2} = 1770$, for $\delta = 3$, $\binom{59}{2} = 1711$ and for $\delta = 4$, $\binom{58}{2} = 1653$.
For $p = 3$, $\Delta = i_3 - i_2 - 1 = 8 - 7 - 1 = 0$.

For $p = 4$, $\Delta = i_4 - i_3 - 1 = 14 - 8 - 1 = 5$ *and for* $\delta = 1$, $\binom{64-8-1}{4-4} = \binom{55}{0} = 1$, *for* $\delta = 2$, $\binom{54}{0} = 1$, *for* $\delta = 3$, $\binom{53}{0} = 1$, *for* $\delta = 4$, $\binom{52}{0} = 1$ *and for* $\delta = 5$, $\binom{51}{0} = 1$.
The sum of all numbers lead to $1 + 46140 = 46141$.

4.1.5 Parallelization

The sequential algorithm *NA* generates a code X_l^j from its preceding code using the function $next''\left(X_l^{j-1}\right)$. A first way for its parallelization consists to launch a thread per tree root. Indeed, the first code X_l of a tree $\mathcal{T}_l^t$ is obtained by $X_l = \left\{T^t, \ldots, T^{t+l-1}\right\}$. However, this approach is inefficient as the workload is not well shared. For example, the first tree has much more codes than the last tree (only one code) and the branch pruning also introduces some unpredictability as the number of skipped codes per tree is unknown. A good parallelization depends also on the computer on which the algorithm runs, e.g. a parallelization with eight threads on a computer with four cores is useless. So, the number of threads will be a parameter of the parallel algorithm *NA*.

The parallel algorithm *NA* is based on the thread pool model. A pool of n threads, n being the number of available processor cores, waits for tasks. When a task needs to be done, a thread is taken from the pool to compute the task. When the task is finished, the thread is returned to the pool. In the algorithm *NA*, tasks are sets of x contiguous codes. This number x is determined by the number of tasks and the number of codes to be tested for a given length l. Code slices in the forest $\mathcal{F}_l$ allows an implementation of this procedure (Figure 4). The number of tasks is also a parameter of the parallel algorithm *NA*. For our benchmark (Section 5), this parameter is set to 512 tasks.

A slice starts at a code number j, spans for the x next codes which can cover several trees or only a portion of one tree, and ends at code $j+x$. Hence, a task is defined by two codes being the inclusive boundaries of a code slice, i.e. X_l^j is the lower bound of the task while X_l^{j+x} is its upper bound. The code X_l^j is built from its number j. Then, the necklace test is performed on X_l^j before calling the function $next''$. This operation is repeated until $next''$ either stops by itself or produce a code greater than X_l^{j+x}. So, the problem to build a code from a number is the inverse operation of calculating the number

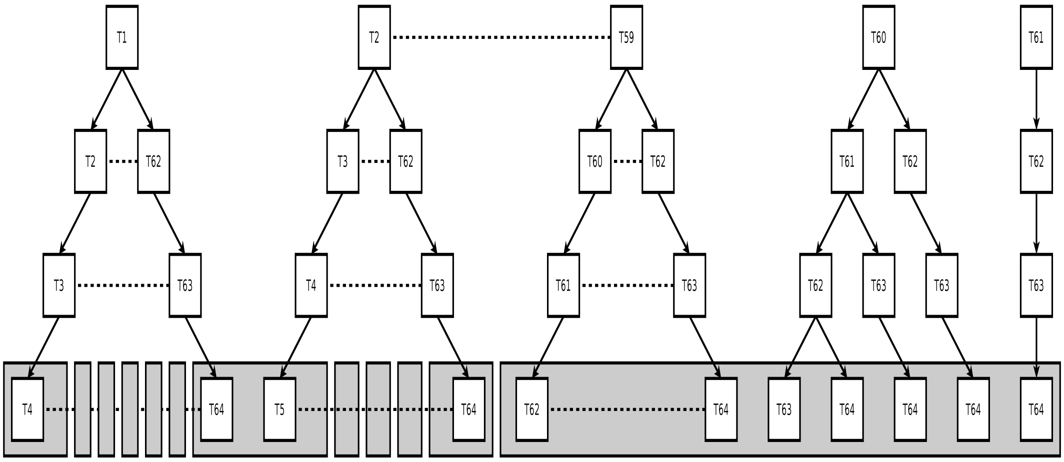

Figure 4: Example of slices in the forest $\mathcal{F}_l$. Slices have equal size in contrast to the trees where the first trees have more codes than the last trees.

of a code (Definition 4.44). Thus, the concepts are very similar to previously.

A code number j, as before, is the number of previous codes plus one. But instead of finding how many codes a previous incomplete branch represent in order to calculate j, we determine if a given incomplete branch $X_{m,l}$ has enough codes so that $X_l^j \in (X_{m,l})^*$, i.e. $j \leq |X_{m,l}|$. Thus, we define two functions: the function *doCode* which encapsulates the function *doCode'*.

Notation 4.46. *For a trinucleotide subcode $X_{m,l}$ and a trinucleotide T^i with $T^i > X_{m,l}(m)$, the concatenation of T^i after $X_{m,l}(m)$ is noted $X_{m+1,l} = X_{m,l} \cdot T^i$ if $m+1 < l$ or $X_l = X_{m,l} \cdot T^i$ if $m+1 = l$.*

Definition 4.47. *For a code length $1 \leq l \leq 20$, a number $1 \leq j_{gap} \leq j_{\max}(l)$, a position parameter $1 \leq p \leq l$, an order parameter $1 \leq i \leq 64$ and a code $X_{p-1,l}$, the recursive function doCode' $\left(j_{gap}, i, X^j_{p-1,l}\right)$ computes the code X_l as follows*

$$
doCode'\left(j_{gap}, i, X^j_{p-1,l}\right) =
$$
$$
\begin{cases}
doCode'\left(j_{gap}-n, i+1, X^j_{p-1,l}\right) \text{ with } n = \binom{64-i}{l-p} \text{ if } j_{gap} > n \\
doCode'\left(j_{gap}, i+1, X^j_{p,l}\right) \text{ with } X^j_{p,l} = X^j_{p-1,l} \cdot T^i \text{ and } n = \binom{64-i}{l-p} \\
\qquad \text{if } j_{gap} \leq n \text{ and } p < l \\
X^j_l \text{ with } X^j_l = X^j_{p-1,l} \cdot T^i \text{ with } n = \binom{64-i}{l-p} \text{ if } j_{gap} = n \text{ and } p = l \\
\text{undefined otherwise}
\end{cases}
$$

Definition 4.48. *For a code length* $1 \leq l \leq 20$ *and a code number* $1 \leq j \leq j_{\max}(l)$*, the function* $doCode(l, j) = doCode'(j, 1, \{\}_{0,l})$ *gives the code* X_l^j*.*

The function $doCode'$ enumerates, counting backward, the possible trinucleotides T^i for a given current position p in order to grow the subcode $X_{p-1,l}^j$. The parameter j_{gap} represents the distance from the desired code X_l^j. Initially, $j_{gap} = j$, i.e. the code X_l^j has a distance j_{gap} from the beginning of the forest $\mathcal{F}_l$. There are three main cases:

1. If the number j_{gap} is greater than the number $|Y_{p,l}|$ of codes in $(Y_{p,l})^*$ with $Y_{p,l} = \left\{X_{p-1,l}, T_p^i\right\}$ then the desired code X_l^j is beyond $Y_{p,l}$, i.e. its trinucleotide $X_l^j(p)$ at position p is greater than T^i. The function recursively calls itself for T^{i+1} at the current position while substracting $|Y_{p,l}|$ to j as codes in $(Y_{p,l})^*$ just have been skipped. Intuitively, we travel in $\mathcal{F}_l$ from left to right, skipping over $(Y_{p,l})^*$ when $X_l^j \notin (Y_{p,l})^*$.

2. If the number j_{gap} is less or equal than the number $|Y_{p,l}|$ of codes in $(Y_{p,l})^*$ with $Y_{p,l} = \left\{X_{p-1,l}, T_p^i\right\}$ then $Y_{p,l}$ is a subcode of X_l^j. The function recursively calls itself in order to compute the trinucleotide at the next position. Intuitively, we travel in $\mathcal{F}_l$ from top to bottom.

3. When the code X_l^j is complete, it is returned by the function $doCode'$.

Example 4.49. *Example 4.45 is used to determine the code* $X_4^{46141} = \{T^2, T^7, T^8, T^{14}\}$ *from the number* $j = 46141$ *(*$l = 4$*). We have* $doCode(4, 46141) = doCode'\left(46141, 1, \{\}_{0,4}\right)$.

As $46141 > \binom{64-1}{4-1} = 39171$ *then* $doCode'\left(6970, 2, \{\}_{0,4}\right)$ $(6970 = 46141 - 39171)$.

As $6970 \leq \binom{64-2}{4-1} = 37820$ *then* $doCode'\left(6970, 3, \{T^2\}_{1,4}\right)$.

As $6970 > \binom{64-3}{4-2} = 1830$ *then* $doCode'\left(5140, 4, \{T^2\}_{1,4}\right)$.

As $5140 > \binom{64-4}{4-2} = 1770$ *then* $doCode'\left(3370, 5, \{T^2\}_{1,4}\right)$.

As $3370 > \binom{64-5}{4-2} = 1711$ *then* $doCode'\left(1659, 6, \{T^2\}_{1,4}\right)$.

As $1659 > \binom{64-6}{4-2} = 1653$ *then* $doCode'\left(6, 7, \{T^2\}_{1,4}\right)$.

As $6 \leq \binom{64-7}{4-2} = 1596$ *then* $doCode'\left(6, 8, \{T^2, T^7\}_{2,4}\right)$.

As $6 \leq \binom{64-8}{4-3} = 56$ *then* $doCode'\left(6, 9, \{T^2, T^7, T^8\}_{3,4}\right)$.

$As\ 6 > \binom{64-9}{4-4} = 1\ then\ doCode'\left(5, 10, \{T^2, T^7, T^8\}_{3,4}\right)$.
$As\ 5 > \binom{64-10}{4-4} = 1\ then\ doCode'\left(4, 11, \{T^2, T^7, T^8\}_{3,4}\right)$.
$As\ 4 > \binom{64-11}{4-4} = 1\ then\ doCode'\left(3, 12, \{T^2, T^7, T^8\}_{3,4}\right)$.
$As\ 3 > \binom{64-12}{4-4} = 1\ then\ doCode'\left(2, 13, \{T^2, T^7, T^8\}_{3,4}\right)$.
$As\ 2 > \binom{64-13}{4-4} = 1\ then\ doCode'\left(1, 14, \{T^2, T^7, T^8\}_{3,4}\right)$.
$As\ 1 \le \binom{64-14}{4-4} = 1\ and\ p = 4 = l\ then\ \{T^2, T^7, T^8\}{\cdot}T^{14} = \{T^2, T^7, T^8, T^{14}\}_4^{46141}$.

4.2 The necklace algorithm NA in two dimensions (conjugate class order)

We extend the previous definitions to a conjugate class order. From Proposition 3.5, the trinucleotide codes are generated by the algorithm NA according to the following partition of $\mathcal{B}_4^3$ into the 20 conjugate classes (Table 1).

Table 1: Trinucleotide partition of $\mathcal{B}_4^3$ into 20 classes of 3 permuted trinucleotides.

	Permutation v		
Class c	1	2	3
1	AAC	ACA	CAA
2	AAG	AGA	GAA
3	AAT	ATA	TAA
4	ACC	CCA	CAC
5	ACG	CGA	GAC
6	ACT	CTA	TAC
7	AGC	GCA	CAG
8	AGG	GGA	GAG
9	AGT	GTA	TAG
10	ATC	TCA	CAT
11	ATG	TGA	GAT
12	ATT	TTA	TAT
13	CCG	CGC	GCC
14	CCT	CTC	TCC
15	CGG	GGC	GCG
16	CGT	GTC	TCG
17	CTG	TGC	GCT
18	CTT	TTC	TCT
19	GGT	GTG	TGG
20	GTT	TTG	TGT

Notation 4.50. *For a class number* $1 \le c \le 20$ *and a permutation number*

$1 \leq v \leq 3$, $\mathcal{B}_4^3(c,v) = T^{c,v}$ is the trinucleotide $T^{c,v}$ belonging to the class c and the permutation v according to Table 1.

Example 4.51. $\mathcal{B}_4^3(2,1) = AAG$.

Remark 4.52. *The symbol v (for "variant") is used for the permutation as the symbol p is already used for "position".*

The previous notation T^i in one dimension (i dimension) is extended to two dimensions (c and v dimensions). This extension does not modify the necklace test which is based on the position of trinucleotides and not on the order of trinucleotides. In fact, the necklace test already relies on $\mathcal{B}_4^3$ as trinucleotide codes without $\{AAA, CCC, GGG, TTT\}$ and without conjugate trinucleotides are prerequisites for the test. However, this extension affects the trinucleotide code generation process and its related formulas. The different functions depending on the i dimension, $1 \leq i \leq 64$, now depend on the c first dimension, $1 \leq c \leq 20$, and on the v second dimension, $1 \leq v \leq 3$.

Definition 4.53. *For two trinucleotides T^{c_1,v_1} and T^{c_2,v_2}, $T^{c_1,v_1} < T^{c_2,v_2}$ if either $c_1 < c_2$ or $c_1 = c_2$ and $v_1 < v_2$.*

Definition 4.54. *A trinucleotide code X_l^j of length $1 \leq l \leq 20$ with $1 \leq j \leq j_{\max}(l) = \binom{20}{l} \times 3^l$ (consequence of Proposition 3.5) is composed of l trinucleotides from distinct classes, i.e.*
$X_l^j = \left\{T_1^{c_1,v_1}, \ldots, T_{p_1}^{c_{p_1},v_{p_1}}, \ldots, T_{p_2}^{c_{p_2},v_{p_2}}, \ldots, T_l^{c_l,v_l}\right\}$ with $1 \leq p_1 < p_2 \leq l$ and $c_{p_1} < c_{p_2}$.

Definition 4.55. *For a code length $1 \leq l \leq 20$, a trinucleotide T^{c_p,v_p} is a limit trinucleotide $T_{\lim}$ in the code $X_l = \left\{T_1^{c_1,v_1}, \ldots, T_p^{c_p,v_p}, \ldots, T_l^{c_l,v_l}\right\}$ at position $1 \leq p \leq l$ if $T^{c_p,v_p} = T^{20+p-l,3}$. Limit trinucleotides only exist in the last column of Table 1.*

Example 4.56. *The code $X_3 = \left\{T_1^{4,1}, T_2^{5,2}, T_3^{6,3}\right\}$ has no limit trinucleotide. The code $Y_3 = \left\{T_1^{4,1}, T_2^{19,3}, T_3^{20,3}\right\}$ has two limit trinucleotides $T_2^{19,3}$ and $T_3^{20,3}$.*

Remark 4.57. *Contrary to the classical lexicographical order, limit trinucleotides are not always contiguous and at the end of the code in the conjugate class, e.g. $Z_3 = \left\{T_1^{4,1}, T_2^{19,3}, T_3^{20,1}\right\}$ is a valid trinucleotide code with a limit trinucleotide $T_2^{19,3}$ and a non-limit trinucleotide $T_3^{20,1}$.*

In the trinucleotide code generation process, only the two functions *push* and *push'* need a redefinition. However, for completeness, all functions are adapted to the two dimensions.

Definition 4.58. *For a trinucleotide $T^{c,v}$, the function $next\left(T^{c,v}\right)$ giving the successor of $T^{c,v}$ is defined by*

$$next\left(T^{c,v}\right)=\begin{cases} T^{c+1,v} & \textit{if } c<20 \textit{ and } v=3 \\ T^{c,v+1} & \textit{if } v<3 \\ \textit{undefined for } T^{20,3} \end{cases}$$

Definition 4.59. *For a trinucleotide $T^{c,v}$, the function $nextClass\left(T^{c,v}\right)$ giving the first permutation of the following class of $T^{c,v}$ is defined by*

$$nextClass\left(T^{c,v}\right)=\begin{cases} T^{c+1,1} & \textit{if } c<20 \\ \textit{undefined for } c=20 \end{cases}$$

Definition 4.60. *For a trinucleotide code X_l^j, the function $pop\left(X_l^j\right)$ returning a couple (subcode, last removed trinucleotide) is defined by*

$$pop\left(X_l^j\right)=\begin{cases} \left(X_{l-1,l}^{j+1},T^{c_l,v_l}\right)=\left(\{T_1,\ldots,T_{l-1}\}_{l-1,l}^{j+1},T^{c_l,v_l}\right) \\ \qquad \textit{if } X_l^j=\{T_1,\ldots,T_{l-1},T_l^{c_l,v_l}\} \textit{ and } T^{c_l,v_l}\neq T_{\lim} \\ \left(X_{p-2,l}^{j+1},T^{c_{p-1},v_{p-1}}\right)=\left(\{T_1,\ldots,T_{p-2}\}_{p-2,l}^{j+1},T^{c_{p-1},v_{p-1}}\right) \\ \qquad \textit{if } X_l^j=\left\{T_1,\ldots,T_{p-2},T_{p-1}^{c_{p-1},v_{p-1}},T_p^{(20-l+p),3},\ldots,T_l^{20,3}\right\} \\ \qquad \textit{with } l-p+1 \textit{ limit trinucleotides } T_{\lim} \end{cases}$$

Definition 4.61. *For a trinucleotide subcode $X_{m,l}^j=\{T_1,\ldots,T_m^{c_m,v_m}\}_{m,l}^j$ and a trinucleotide $T^{c_{last},v_{last}}\geq X_{m,l}^j(m)$, $1\leq c_{last}\leq 20+m-l$ and $1\leq$*

$v_{last} \leq 3$, *the function push* $\left(X^j_{m,l}, T^{c_{last},v_{last}}\right)$ *is defined by*

$$push\left(X^j_{m,l}, T^{c_{last},v_{last}}\right) = \begin{cases} \left(X^j_{m+1,l}, T\right) = \left(\{T_1, \ldots, T_m^{c_m,v_m}, T_{m+1}\}^j_{m+1,l}, T\right) \\ \quad \textit{with } T = next\left(T^{c_{last},v_{last}}\right) \\ \quad \textit{if } m < l-1 \textit{ and } c_m \neq c_{last} \\ \left(X^j_{m+1,l}, T\right) = \left(\{T_1, \ldots, T_m^{c_m,v_m}, T_{m+1}\}^j_{m+1,l}, T\right) \\ \quad \textit{with } T = nextClass\left(T^{c_{last},v_{last}}\right) \\ \quad \textit{if } m < l-1 \textit{ and } c_m = c_{last} \\ \left(X^j_{l}, T\right) = \left(\{T_1, \ldots, T_m^{c_m,v_m}, T_l\}^j_{l}, T\right) \\ \quad \textit{with } T = next\left(T^{c_{last},v_{last}}\right) \\ \quad \textit{if } m = l-1 \textit{ and } c_m \neq c_{last} \\ \left(X^j_{l}, T\right) = \left(\{T_1, \ldots, T_m^{c_m,v_m}, T_l\}^j_{l}, T\right) \\ \quad \textit{with } T = nextClass\left(T^{c_{last},v_{last}}\right) \\ \quad \textit{if } m = l-1 \textit{ and } c_m = c_{last} \end{cases}$$

The new function *push* has two times more cases as it must consider the second dimension. If the last trinucleotide of the subcode $X^j_{m,l}$ is different from $T^{c_{last},v_{last}}$ then the code X^{j-1}_l has just been removed and the successor $next\left(T^{c_{last},v_{last}}\right)$ of $T^{c_{last},v_{last}}$ must be pushed. Otherwise, the subcode $X^j_{m,l}$ must be completed by the trinucleotides $T = nextClass\left(T^{c_{last},v_{last}}\right)$ from the classes after c_{last} (Figure 5).

The function $next\left(X^j_l\right)$ is unmodified and based on Definition 4.19.

The functions for branch pruning are quite similar.

Definition 4.62. *For a trinucleotide subcode* $X^j_{m,l}$, *the function* $next'\left(X^j_{m,l}\right)$ *is defined by*

$$next'\left(X^j_{m,l}\right) = push'^*\left(pop'\left(X^j_{m,l}\right)\right) = \left(X^{j_{next}}_l, T^{c_{last},v_{last}}\right)$$

The function $next''\left(X^j_l\right)$ is unchanged and based on Definition 4.43.

Definition 4.63. *For a trinucleotide subcode* $X^j_{m,l} = \{T_1, \ldots, T_m^{c_m,v_m}\}^j_{m,l}$, *the function* $pop'\left(X^j_{m,l}\right)$ *returning a couple (subcode, last removed trinucleotide)*

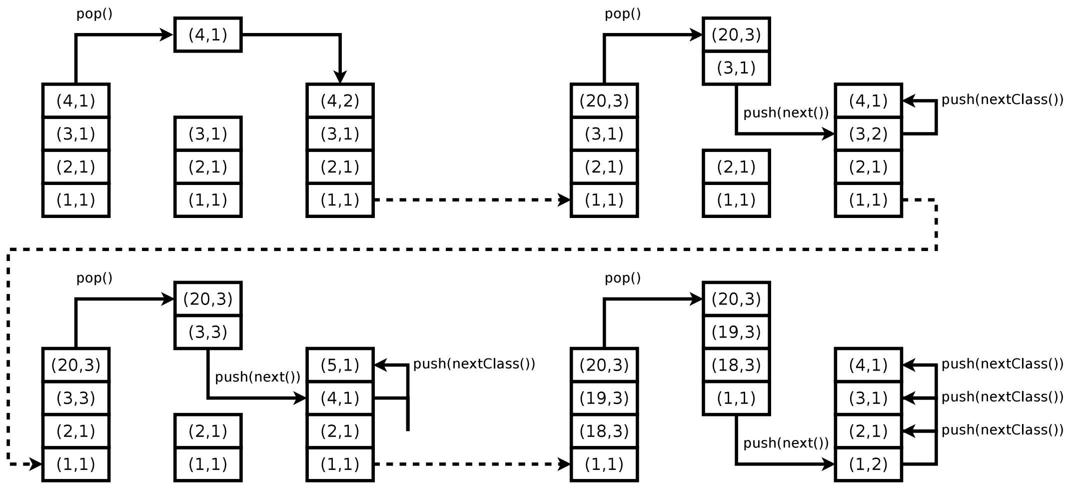

Figure 5: Graphical representation of the generation of codes from $\{T^{1,1}, T^{2,1}, T^{3,1}, T^{4,1}\}_4$ to $\{T^{1,2}, T^{2,1}, T^{3,1}, T^{4,1}\}_4$ with the functions *pop* and *push* in $\mathcal{B}_4^3$. Codes are represented vertically from bottom to top, with trinucleotides (c, v) (see Table 1).

is defined by

$$
pop'\left(X_{m,l}^{j}\right) =
$$
$$
\begin{cases}
\left(X_{m-1,l}^{j+1}, T^{c_m,v_m}\right) = \left(\{T_1, \ldots, T_{m-1}\}_{m-1,l}^{j+1}, T^{c_m,v_m}\right) \\
\qquad \textit{if } X_{m,l}^{j} = \{T_1, \ldots, T_{m-1}, T^{c_m,v_m}\} \ \textit{ and } T^{c_m,v_m} \neq T_{\lim} \\
\left(X_{p-2,l}^{j+1}, T^{c_{p-1},v_{p-1}}\right) = \left(\{T_1, \ldots, T_{p-2}\}_{p-2,l}^{j+1}, T^{c_{p-1},v_{p-1}}\right) \\
\qquad \textit{if } X_l^j = \left\{T_1, \ldots, T_{p-2}, T_{p-1}^{c_{p-1},v_{p-1}}, T_p^{(20-l+p),3}, \ldots, T_m^{(20-l+m),3}\right\} \\
\qquad \textit{with } m-p+1 \textit{ limit trinucleotides } T_{\lim}
\end{cases}
$$

Definition 4.64. *For a trinucleotide subcode* $X_{m,l}^{j} = \{T_1, \ldots, T_m^{c_m,v_m}\}_{m,l}^{j}$ *and a trinucleotide* $T^{c_{last},v_{last}} \geq X_{m,l}^{j}(m)$, $1 \leq c_{last} \leq 20 + m - l$, *the function* $push'\left(X_{m,l}^{j}, T^{c_{last},v_{last}}\right)$ *is defined by*

$$
push'\left(X_{m,l}^{j}, T^{c_{last},v_{last}}\right) = \begin{cases}
push\left(X_{m,l}^{j}, T^{c_{last},v_{last}}\right) & \textit{if } X_{m,l}^{j} \textit{ is circular} \\
next'\left(X_{m,l}^{j}\right) & \textit{otherwise}
\end{cases}
$$

The functions $number\left(X_l\right)$ and $doCode'\left(j\right)$ also need some extension. The $number\left(X_l\right)$ depends on the trinucleotide position in the code and the trinucleotide order which has now two dimensions. For the class dimension, the number of codes is counted in previous classes and for the permutation dimension, the number of codes is counted in the current class but in the previous permutation number. The definition of $|X_{m,l}|$ must also be adapted for the formula *number* (X_l).

Definition 4.65. *The number of complete trinucleotide codes sharing a subcode* $X_{m,l}$ *is* $|X_{m,l}| = 3^{l-m}\binom{20-c_m}{l-m}$ *with* $X_{m,l}(m) = T^{c_m,v_m}$.

Example 4.66. *With the code length* $l = 20$ *and* $m = 19$, $|X_{m,l}| = 3^{20-19}\binom{20-19}{20-19} = 3$. *Indeed, if* $l = 20$, *all classes are used, i.e. for* $m = 19$, $c_m = 19$ *and the next trinucleotide is in the* $20th$ *class with three permutation choices (Table 1). Note also that in the particular case with* $l = 20$, *as all classes are used,* $p = c_p$ *and* $T = X_{20}(p)$ *for all positions* $1 \le p \le 20$.

$|X_{m,l}|$ gives the number of codes sharing a subcode $X_{m,l}$ whose class of the last trinucleotide T^{c_m,v_m} is c_m, i.e. the number of codes sharing $X_{m,l}$ per permutation of c_m. It does not consider the three permutations of c_m itself. So, the total number of codes sharing $X_{m,l}$ with $X_{m,l}(m) = T^{c_m,v_m}$ for $1 \le v_m \le 3$ is obtained by $3|X_{m,l}|$.

Definition 4.67. *For a code* $X_l^j = \{T_1^{c_1,v_1}, \dots, T_l^{c_l,v_l}\}_l^j$, *its number* j *is defined by*

$$
\begin{aligned}
j &= number\,(X_l) \\
&= 1 + \sum_{p=1}^{l}\left[3^{l-p+1}\sum_{\delta=1}^{\Delta}\binom{20-c_{p-1}-\delta}{l-p} + 3^{l-p}(v_p-1)\binom{20-c_p}{l-p}\right]
\end{aligned}
\tag{2}
$$

with $\Delta = c_p - c_{p-1} - 1$, $X_l(p) = T_p^{c_p,v_p}$, $X_l(p-1) = T_{p-1}^{c_{p-1},v_{p-1}}$ *and* $X_l(0) = T_0^{0,0}$, *i.e.* $c_0 = 0$.

The 1st term $3^{l-p+1}\sum_{\delta=1}^{\Delta}\binom{20-c_{p-1}-\delta}{l-p}$ is associated to the code computation in the trinucleotide classes and is equal to $3\sum_{\delta=1}^{\Delta}|X_{p,l}|$ with $X_{p,l} = \left\{T_1^{c_1,v_1}, \dots, T_{p-1}^{c_{p-1},v_{p-1}}, T_p^{c_{p-1}+\delta,v_p}\right\}$. The 2nd term $3^{l-p}(v_p-1)\binom{20-c_p}{l-p}$ is related to the code computation in the previous permutations of the current trinucleotide class c_p and is equal to $(v_p-1)|X_{p,l}|$.

Example 4.68. *For the code* $X_3^j = \{T^{1,2}, T^{3,2}, T^{5,3}\}_3^j = \{ACA, ATA, GAC\}_3^j$, $j = 1758$, *i.e.* $X_3^{1758} = \{ACA, ATA, GAC\}_3^{1758}$. *Indeed, by computing Formula 2, we have the following partial results.*

For $p = 1$, $\Delta = c_1 - c_0 - 1 = 1 - 0 - 1 = 0$, $v_1 - 1 = 2 - 1 = 1$, $3^{3-1}(1)\binom{20-1}{3-1} = 9\binom{19}{2} = 1539$.
For $p = 2$, $\Delta = c_2 - c_1 - 1 = 3 - 1 - 1 = 1$, $v_2 - 1 = 2 - 1 = 1$ *and for* $\delta = 1$, $3^{3-2+1}\binom{20-1-1}{3-2} = 9\binom{18}{1} = 162$, *and then* $3^{3-2}(1)\binom{20-3}{3-2} = 3\binom{17}{1} = 51$.
For $p = 3$, $\Delta = c_3 - c_2 - 1 = 5 - 3 - 1 = 1$, $v_3 - 1 = 3 - 1 = 2$ *and for* $\delta = 1$, $3^{3-3+1}\binom{20-3-1}{3-3} = 3\binom{16}{0} = 3$, *and then* $3^{3-3}(2)\binom{20-5}{3-3} = 2\binom{15}{0} = 2$.
The sum of all numbers lead to $1 + 1757 = 1758$.

For the inverse formula, the function *doCode* is unmodified and based on Definition 4.48. The function *doCode*$'$ must be adapted to the two dimensions of class and permutation.

Definition 4.69. *For a code length* $1 \leq l \leq 20$, *a code number* $1 \leq j_{gap} \leq j_{\max}(l)$, *a position parameter* $1 \leq p \leq l$, *a class number* $1 \leq c \leq 20$ *and a code* $X_{p-1,l}$, *the function doCode*$'\left(j_{gap}, c, X^j_{p-1,l}\right)$ *computes the code* X_l

$$doCode'\left(j_{gap}, c, X^j_{p-1,l}\right) = \begin{cases} doCode'\left(j_{gap} - 3n, c+1, X_{p-1,l}\right) & \\ \qquad with\ n = 3^{l-p}\binom{20-c}{l-p}\ if\ j_{gap} > 3n & \\ doCode'\left(j_{gap} - (v-1)\,n, c+1, X_{p,l}\right) & \\ \qquad with\ X_{p,l} = X_{p-1,l} \cdot T^{c,v},\ v = \left\lceil \frac{j_{gap}}{n} \right\rceil & \\ \qquad and\ n = 3^{l-p}\binom{20-c}{l-p}\ if\ j_{gap} \leq 3n\ and\ p < l & \\ X_l\ with\ X_l = X_{p-1,l} \cdot T^{c,v},\ v = \left\lceil \frac{j_{gap}}{n} \right\rceil & \\ \qquad and\ n = 3^{l-p}\binom{20-c}{l-p}\ if\ j_{gap} \leq 3n\ and\ p = l & \\ undefined\ otherwise & \end{cases}$$

The principle is similar to the one dimension case. The current subcode $X_{p,l}$ must be shared by enough complete codes so that the code X_p belongs to $(X_{p,l})^*$. However, the number $|X_{p,l}| = n = 3^{l-p}\binom{20-c}{l-p}$ of complete codes is computed for a class number and multiplied by 3 for considering the 3 permutations of the current class c. If the current class c is the one that must be considered, the permutation number is computed by $v_p = \left\lceil \frac{j_{gap}}{n} \right\rceil$, i.e. the ceiling of the number j_{gap} of the code divided by the number n of codes per permutation of the current class c. When a trinucleotide is found, *doCode*$'\left(j_{gap} - (v-1)\,n, c+1, X_{p,l}\right)$ eliminates the $(v-1)\,n$ previous permutations. Note that when $v = 1$, *doCode*$'\left(j_{gap}, c+1, X_{p,l}\right)$ matches the one dimension Definition 4.47 as no permutation is counted.

Example 4.70. *Example 4.68 is used to determine the code* $X_3^{1758} = \{T^{1,2}, T^{3,2}, T^{5,3}\}$ *from the number* $j = 1758$ *(*$l = 3$*). We have* $doCode\,(3, 1758) = doCode'\left(1758, 1, \{\}_{0,3}\right)$.
As $1758 \leq 3n = 3^{3-1}\binom{20-1}{3-1} = 1539$ *and* $v = \left\lceil \frac{1758}{1359} \right\rceil = 2$, *then*
$doCode'\left(219, 2, \{T^{1,2}\}_{1,3}\right)$ $(219 = 1758 - (2-1) \times 1539)$.
As $219 > 3n = 3^{3-2}\binom{20-2}{3-2} = 54$, *then*
$doCode'\left(57, 3, \{T^{1,2}\}_{1,3}\right)$ $(57 = 219 - 3 \times 54)$.
As $57 \leq 3n = 3^{3-2}\binom{20-3}{3-2} = 51$ *and* $v = \left\lceil \frac{57}{51} \right\rceil = 2$, *then*
$doCode'\left(6, 4, \{T^{1,2}, T^{3,2}\}_{2,3}\right)$ $(6 = 57 - (2-1) \times 51)$.
As $6 > 3n = 3^{3-3}\binom{20-4}{3-3} = 1$, *then* $doCode'\left(3, 5, \{T^{1,2}, T^{3,2}\}_{2,3}\right)$ $(3 = 6 - 3 \times 1)$.
As $3 \leq 3n = 3^{3-3}\binom{20-5}{3-3} = 1$, $v = \left\lceil \frac{3}{1} \right\rceil = 3$ *and* $p = 3 = l$ *then* $\{T^{1,2}, T^{3,2}\}_{2,3} \cdot T^{5,3} = \{T^{1,2}, T^{3,2}, T^{5,3}\}_3^{1758}$.

All formulas being defined for the conjugation classes on $\mathcal{B}_4^3$, the algorithm *NA* is complete. It generates, by construction, trinucleotide codes without $\{AAA, CCC, GGG, TTT\}$ and without conjugate trinucleotides. It uses branch pruning while generating trinucleotide codes. It can recover the number j of a code from its trinucleotides and can generate a trinucleotide code from a number j and a length, allowing to define lower and upper boundaries for a parallel program.

4.3 Implementation hints

We now give some implementation hints for coding the necklace algorithm *NA* in the Java programming language.

4.3.1 Trinucleotide representation

A letter on $\mathcal{A}_4$ can be coded on two bits and a trinucleotide, on six bits. The permutation number can also be coded on two bits and the class number on five bits. Thus, a Java 16 bits integer can represent a trinucleotide with the following conventions (Figure 6):

- The first 6 bits represent a trinucleotide which is masked with the hexadecimal value `0X3F`.

  - The first letter is masked by the hexadecimal value `0X30`.
  - The second letter is masked by the hexadecimal value `0XC`.
  - The third letter is masked by the hexadecimal value `0X3`.

- The 2 following bits represent the trinucleotide permutation number which is masked with the hexadecimal value `0XC0`.

- The 6 following bits represent the trinucleotide class number (even if 5 bits are enough) which is masked with the hexadecimal value `0X3F00`.

The letter A is coded by the hexadecimal value `0X0`, the letter C by `0X1`, the letter G by `0X2` and the letter T by `0X3`.

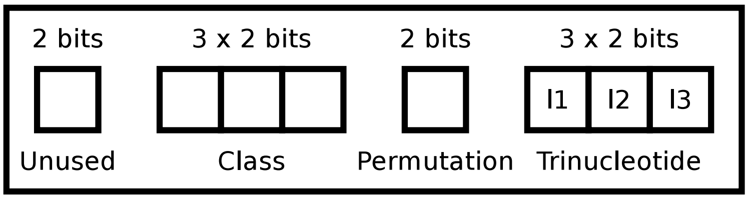

Figure 6: Representation of a trinucleotide on a 16 bits java integer. Each square stands for 2 bits. $l1$, $l2$ and $l3$ are the letters of the trinucleotide.

Such a representation is efficient in time and space because it only uses one integer which is a basic type in Java. Indeed, there is no dynamic memory allocation, i.e. no call to any *new* object creation method.

4.3.2 Trinucleotide code representation

Trinucleotide codes are composed of trinucleotides which are implemented by integers. A trinucleotide code of length l is allocated with an array of l integers. This array is then used as a stack by the functions *pop* and *push*, using an integer *count* to keep track of the current length of the code. The same array is used for all the codes.

One of the most used operations on the code is the inclusion test which tests if a trinucleotide T belongs or not to a code X. Considering the maximum length of the code, i.e. $l = 20$, a linear time search could be acceptable. However, this operation is performed billions of times. Hence, a constant time search is better for the inclusion test. For that purpose, the code contains a

64 boolean array $BArray$. The boolean $BArray[i]$ (0-indexed) is set to true when a trinucleotide is pushed in the code, $0 \leq i \leq 63$ being the value formed by the six lower bits of the trinucleotide code representation.

For the necklace test, a matrix $\mathcal{M}$ is used. This matrix is updated by the function *push*. Indeed, when a trinucleotide is pushed, the top row and top column are updated. As the necklace test only considers the number *count* of trinucleotides in the code when it is executed, there is no need to erase any row or column when popping. They are simply ignored and overwritten later by the function *push*. So, the matrix $\mathcal{M}$ evolves along with the code, and as for the code, it is preallocated according to the length l.

4.3.3 Sequential necklace test

The necklace test is coded sequentially with four imbricated loops instead of recursive calls. However, note that the 2nd case in the function $chain'$ uses a "or" clause allowing to stop immediately (without calculating all chains) when a closed chain is found. The implementation uses that fact to bail out as soon as possible. The same remark applies to the function *test*.

Another optimization is added to the necklace test. If the trinucleotide $T_{p_1,p_2} = d_{p_1}l_{p_2} \in X_l^j$ participates to a necklace then there is a trinucleotide $T_{p_2,p_3} = d_{p_2}l_{p_3} \in X_l^j$. Thus, if d_{p_2} is not a diletter prefix of a trinucleotide in X_l^j then the necklace search can be stopped earlier. So, an additional inclusion test based on diletters of the code, instead on trinucleotides, is performed with a new 16 integer array $IArray$. When a trinucleotide $T = dl$ is pushed in the code, $IArray[d]$ is incremented. Conversely, it is decremented when a trinucleotide dl is removed from the code.

A last optimization is added as the necklace is sequentially implemented. When the function $row\left(Current, X_l^j\right)$ (Definition 4.34) searches for a trinucleotide in the $Current$ row of matrix $\mathcal{M}\left(X_l^j\right)$, it searchs for all trinucleotides belonging to X_l^j. However, this search yields unnecessary tests. For example, if the first trinucleotide is d_2l_4 in $\mathcal{M}\left(X_l^j\right)$, the necklace search can begin with the trinucleotide d_4l_3 as the trinucleotides d_4l_1 and d_4l_2 were already tested with the previous search of chains d_1l_4, d_4l_1 and d_2l_4, d_4l_2, i.e. two necklaces $3LDCCN$ would have been found in the cases $d_4l_1 \in X_l^j$ and $d_4l_2 \in X_l^j$. Hence, the search of trinucleotides T_{p_2,p_3} and T_{p_3,p_4}, i.e. in the p_2th and p_3th rows of

$\mathcal{M}\left(X_l^j\right)$, can begin at the column p_1+1 of $\mathcal{M}\left(X_l^j\right)$ as the trinucleotides $T_{p_2,q}$ and $T_{p_3,q}$ for $q<p_1+1$ were already tested.

5 Benchmark

The algorithm *NA* was executed on a quad core processor, precisely an Intel Core i7 K875 running at 2.93 Ghz (processor launched in 2010). The branch pruning necklace test in the function *push* can be tuned to search only for some chain lengths. The effect of the branch punning method developed here is evaluated with the three possible necklace tests: $3LDCCN$ which searches only for chains of length 2, $\{3,4\}\,LDCCN$ which search for chains of lengths 2 and 3 and the complete test $\{3,4,5\}\,LDCCN$ which searches for chains of length 2, 3 and 4.

Table 2 shows the effect of branch pruning on the number of generated codes for the lengths $l=1,\ldots,20$. For each length, the first column gives the number $nbCode\,(l)=\binom{20}{l}\times 3^l$ of trinucleotide codes that would have been generated by the algorithm *NA* without branch pruning. A necklace test should have been necessary on each code in order to decide if a given trinucleotide code is circular or not. The last column presents the number of circular trinucleotide codes per length (identifical to Table 1 in [18]). The 2nd, 3rd and 4th columns give the number of codes generating by the algorithm *NA* using branch pruning.

Table 2 shows that the branch pruning method is very efficient. Indeed, the number of generated codes by the three configurations $3LDCCN$, $\{3,4\}\,LDCCN$ and $\{3,4,5\}\,LDCCN$ is very close to the number of circular trinucleotide codes. Most of the codes are already pruned by the $3LDCCN$ configuration, the $\{3,4\}\,LDCCN$ and $\{3,4,5\}\,LDCCN$ configurations only making a little improvement. The $3LDCCN$ configuration eliminates 923,207, 967,450 codes in total (for all lengths) where the maximum of eliminated codes for a given length is 194,470,034,181 at $l=15$, i.e. 21% of total.

Figure 7 associated to Table 2 gives a graphical representation of the percentage of generated codes by the three configurations $3LDCCN$, $\{3,4\}\,LDCCN$ and $\{3,4,5\}\,LDCCN$ compared to the number $nbCode\,(l)=\binom{20}{l}\times 3^l$ of codes. The percentage of circular codes is also plotted for reference.

Table 2: Number of generated trinucleotide codes per code length l without branch pruning, with branch pruning according to the three configurations $3LDCCN$, $\{3,4\}\,LDCCN$ and $\{3,4,5\}\,LDCCN$, and the number of trinucleotide circular codes (identical to Table 1 in [18]).

l	$\binom{20}{l} \times 3^l$	$3LDCCN$	$\{3,4\}\,LDCCN$	$\{3,4,5\}\,LDCCN$	Circular codes
1	60	60	60	60	60
2	1,710	1,710	1,710	1,710	1,704
3	30,780	30,693	30,693	30,693	30,432
4	392,445	387,468	387,468	387,468	382,164
5	3,767,472	3,658,512	3,658,512	3,658,449	3,568,212
6	28,256,040	26,638,389	26,629,337	26,627,573	25,507,512
7	169,536,240	152,130,729	151,843,008	151,817,298	141,639,780
8	826,489,170	686,579,037	682,549,435	682,313,431	614,568,102
9	3,305,956,680	2,452,006,929	2,419,691,796	2,418,260,871	2,086,742,208
10	10,909,657,044	6,911,500,197	6,747,894,348	6,742,190,448	5,542,646,244
11	29,753,610,120	15,297,984,089	14,748,959,697	14,734,064,286	11,503,061,124
12	66,945,622,770	26,413,731,079	25,158,454,482	25,132,808,781	18,615,667,124
13	123,591,918,960	35,295,247,416	33,310,067,612	33,280,806,827	23,403,485,556
14	185,387,878,440	36,144,717,927	33,961,014,334	33,938,995,390	22,700,634,924
15	222,465,454,128	27,995,419,947	26,330,532,852	26,319,844,311	16,787,523,072
16	208,561,363,245	16,081,427,528	15,214,026,244	15,210,848,227	9,279,022,320
17	147,219,785,820	6,637,219,234	6,337,298,075	6,336,779,999	3,708,717,048
18	73,609,892,910	1,861,836,717	1,796,714,409	1,796,678,841	1,012,099,740
19	23,245,229,340	318,131,676	310,192,548	310,192,140	168,726,792
20	3,486,784,401	25,010,988	24,603,481	24,603,481	12,964,440
Total	1,099,511,627,775	176,303,660,325	167,224,550,101	167,110,910,284	115,606,988,558

For example, for $l = 20$, only $\frac{25{,}010{,}988}{3{,}486{,}784{,}401} \approx 1\%$ of codes are generated with $3LDCCN$. The running time is $15,550$s, i.e. 4h19mn, for generating the 176,303,660,325 codes with $3LDCCN$, $15,825$s, i.e. 4h24mn, for generating the 167,224,550,101 codes with $\{3,4\}\,LDCCN$ and $16,249$s, i.e. 4h31mn, for generating the 167,110,910,284 codes with $\{3,4,5\}\,LDCCN$.

6 Conclusion

We have proposed a necklace algorithm NA, unique in its class, to determine the growth function of trinucleotide circular codes. It involves several computer techniques based on a generated trinucleotide matrix, branch pruning, parallelization and different implementation hints. We are currently trying to determine the growth function of tetranucleotides, i.e. words of 4 letters on

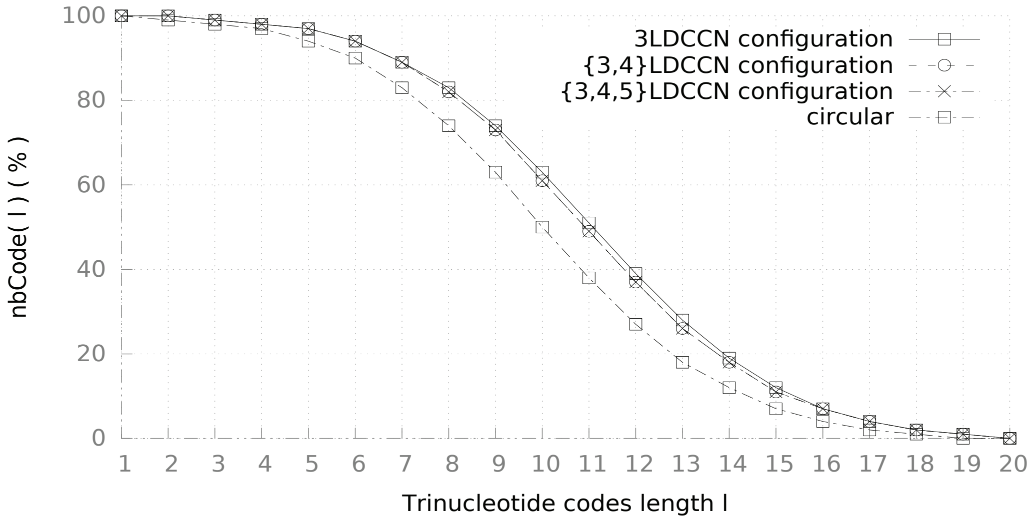

Figure 7: Percentage of generated codes by the three configurations $3LDCCN$, $\{3,4\}\,LDCCN$ and $\{3,4,5\}\,LDCCN$ and circular codes compared to the number $nbCode\,(l) = \binom{20}{l} \times 3^{l}$ of codes.

a 4-letter alphabet, by extending the necklace algorithm NA.

Acknowledgements. We thank Prof. Giuseppe Pirillo for his advices.

References

[1] D.G. Arquès, C.J. Michel, A complementary circular code in the protein coding genes, *J. Theor. Biol.*, **182**, (1996), 45-58.

[2] F. Bassino, Generating function of circular codes, *Adv. Appl. Math.*, **22**, (1999), 1-24.

[3] M.-P. Béal, J. Senellart, On the bound of the synchronization delay of a local automaton, *Theoret. Comput. Science*, **205**, (1998), 297-306.

[4] J. Berstel, D. Perrin, *Theory of Codes*, Vol 117 of *Pure and Applied Mathematics*, Academic Press, London, UK, 1985.

[5] L. Bussoli, C.J. Michel, G. Pirillo, On some forbidden configurations for self-complementary trinucleotide circular codes, *J. Algebra Number Theory Academia*, **2**, (2011), 223-232.

[6] L. Bussoli, C.J. Michel, G. Pirillo, On conjugation partitions of sets of trinucleotides, *Applied Math.*, **3**, (2012), 107-112.

[7] F.H.C. Crick, J.S. Griffith, L.E. Orgel, Codes without commas, *Proc. Natl. Acad. Sci. USA*, **43**, (1957), 416-421.

[8] G. Frey, C.J. Michel, Circular codes in archaeal genomes, *J. Theor. Biol.*, **223**, (2003), 413-431.

[9] G. Frey, C.J. Michel, Identification of circular codes in bacterial genomes and their use in a factorization method for retrieving the reading frames of genes, *Comput. Biol. Chem.*, **30**, (2006), 87-101.

[10] S.W. Golomb, B. Gordon, L.R. Welch, Comma-free codes, *Canad. J. Math.*, **10**, (1958), 202-209.

[11] S.W. Golomb, L.R. Welch, M. Delbrück, Construction and properties of comma-free codes, *Biologiske Meddel Danske Vidensk Selsk*, **23**, (1958), 1-34.

[12] S. Giannerini, D.L. Gonzalez, R. Rosa, DNA, dichotomic classes and frame synchronization: a quasi-crystal framework, *Phil. Trans. R. Soc. A*, **370**, (2012), 2987-3006.

[13] D.L. Gonzalez, S. Giannerini, R. Rosa, Circular codes revisited: a statistical approach, *J. Theor. Biol.*, **275**, (2011), 21-28.

[14] R. Jolivet, F. Rothen, Peculiar symmetry of DNA sequences and evidence suggesting its evolutionary origin in a primeval genetic code, *First European Workshop in Exo-/astro-biology*, Eds.: P. Ehrenfreund, O. Angerer, B. Battrick, Noordwijk, The Netherlands, **496**, (2001), 173-176.

[15] A.J. Koch, J. Lehman, About a symmetry of the genetic code, *J. Theor. Biol.*, **189**, (1997), 171-174.

[16] J.-L. Lassez, Circular codes and synchronization, *Int. J. Computer and Information Sciences*, **5**, (1976), 201-208.

[17] J.-L. Lassez, R.A. Rossi, A.E. Bernal, Crick's hypothesis revisited: the existence of a universal coding frame, *Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops/Symposia* (AINAW '07), **2**, (2007), 745-751.

[18] C.J. Michel, G. Pirillo, Identification of all trinucleotide circular codes, *Comput. Biol. Chem.*, **34**, (2010), 122-125.

[19] C.J. Michel, G. Pirillo, Strong trinucleotide circular codes, *Int. J. Combinatorics*, **2011**, Article ID 659567, (2011), 1-14.

[20] C.J. Michel, G. Pirillo, M.A. Pirillo, Varieties of comma-free codes, *Computers and Mathematics with Applications*, **55**, (2008), 989-996.

[21] C.J. Michel, G. Pirillo, M.A. Pirillo, A relation between trinucleotide comma-free codes and trinucleotide circular codes, *Theoret. Comput. Science*, **401**, (2008), 17-25.

[22] C.J. Michel, G. Pirillo, M.A. Pirillo, A classification of 20-trinucleotide circular codes, *Information and Computation*, **212**, (2012), 55-63.

[23] G. Pirillo, *A characterization for a set of trinucleotides to be a circular code. Determinism, Holism and Complexity*, C. Pellegrini, P. Cerrai, P. Freguglia et al., Eds., Kluwer, Boston, Mass, USA, 2003.

[24] G. Pirillo, A hierarchy for circular codes, *Theoretical Informatics and Applications*, **42**, (2008), 717-728.

[25] G. Pirillo, Some remarks on prefix and suffix codes, *Pure Mathematics and Applications*, **19**, (2008), 53-59.

[26] G. Pirillo, Non sharing border codes, *Advances in Applied Mathematics*, **3**, (2010), 215-223.

[27] G. Pirillo, M.A. Pirillo, Growth function of self-complementary circular codes, *Biology Forum*, **98**, (2005), 97-110.

[28] H. Seligmann, Putative mitochondrial polypeptides coded by expanded quadruplet codons, decoded by antisense tRNAs with unusual anticodons, *Biosystems*, **110**, (2012), 84-106.

[29] N. Štambuk, On circular coding properties of gene and protein sequences, *Croatica Chemica Acta*, **72**, (1999), 999-1008.