Available at WWW.MATHEMATICSWEB.ORG POWERED BY SCIENCE DIRECT®

Applied Numerical Mathematics 46 (2003) 295–318

APPLIED NUMERICAL MATHEMATICS

www.elsevier.com/locate/apnum

# Statistical properties and implementation of aperiodic pseudorandom number generators ☆

Louis-Sébastien Guimond, Jan Patera *, Jiří Patera

*Centre de Recherches Mathématiques, Université de Montréal, C.P. 6128, succ. Centre-Ville, Montréal, PQ, Canada H3C-3J7*

**Abstract**

We discuss the use of quasicrystals in designing deterministic aperiodic pseudorandom number generators. The proposed scheme uses quasicrystals to combine two or three periodic number sequences. We prove that there exists an infinite class of quasicrystals for which the combination scheme (using any nontrivial periodic sequences) produces pseudorandom sequences having no lattice structure. We give empirical results when quasicrystals are used to combine linear congruential generators. Finally, we describe the implementation of two methods for generating quasicrystal points and discuss their respective computational complexities.

## Introduction

Pseudorandom number generators (PRNGs) are designed to have specific statistical properties of independent and identically distributed random variables uniformly distributed over the interval $[0,1)$ (i.i.d. $U[0,1)$). However, the determinism of the generators ensures that the PRNGs have defects and therefore motivates the design of families of generators with distinct statistical properties. In this paper, we consider the aperiodic pseudorandom number generators (APRNGs) whose design was introduced in [6]. Their novelty consists in the use of *quasicrystals* to combine several periodic sequences resulting in an aperiodic number sequence.

The use of aperiodic sequences in the design of PRNGs has already been suggested. For example, the truncated numbers given by $n\theta$ mod 1, where $\theta$ is irrational, are considered in [20] as a quasirandom

☆ This work was supported by the Bell Canada University Laboratories, NSERC of Canada and FCAR of Québec.

* Corresponding author. Current address: Department of Mathematics, Faculty of Nuclear Science and Physical Engineering, Czech Technical University, Trojanova 13, Praha 2, 120 00, Czech Republic

*E-mail addresses:* pateraj@crm.umontreal.ca, patera@km1.fjfi.cvut.cz (Jan Patera), patera@crm.umontreal.ca (Jiří Patera).

0168-9274/03/$30.00 © 2003 IMACS. Published by Elsevier B.V. All rights reserved.
doi:10.1016/S0168-9274(03)00036-9

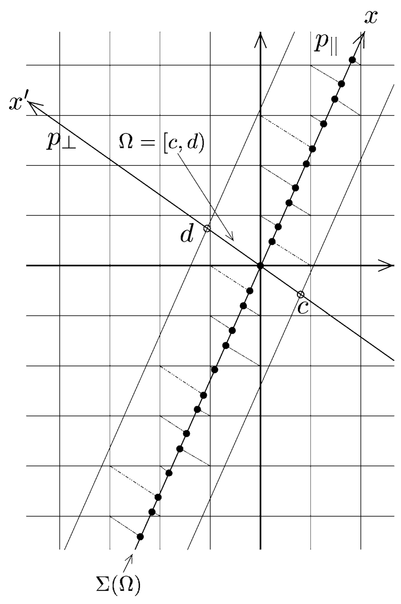


Fig. 1. Cut and project scheme.

sequence with uniform distribution. Another example is given by [19] which uses irrational rotations to generate aperiodic pseudorandom sequences. However, these generators are numeric and may involve rounding errors (this depends on the generator precision and the length of the generated sequence). Hence one of the novelties introduced in the APRNG is to use specifice aperiodic sequences (quasicrystals) which can be generated symbolically and thus exactly, and this even though they are defined using an irrational number.

Most PRNGs have a lattice structure, therefore their parameters must be chosen so as to ensure that their lattice structure is acceptable. (The lattice structure of a pseudorandom sequence may be quantified using the spectral test, see, for example, [9], or the lattice test, see, for example, [17].) The design of PRNGs with no lattice structure (at least in small dimensions) has been proposed. For example, the Inversive Congruential Generators introduced in [4] (see also [8]) have no lattice structure (in all dimensions smaller than $D$, where $D$ depends on the parameters of the generator). In this paper we prove the theoretical results announced in [6], namely that APRNGs have no lattice structure (in any dimension) and we show that they are efficiently generated. We also present some empirical testing results.

The design of the APRNGs involves quasicrystals. These are point sets obtained as the projection of subsets of the $\mathbb{Z}^2$-lattice on a suitably oriented straight line (see Fig. 1). They are geometrically aperiodic and have been thoroughly studied for the past fifteen years as they are used to describe the aperiodic crystalline configuration of physical quasicrystals (see, for example, [1,12,14,13]).

In generic quasicrystals there exist three possible distances between any two adjacent points; in some limit cases there exist only two possible distances. Hence, we say that a quasicrystal generates an aperiodic covering or *tiling* of the real line whose tile types correspond to the possible distances between adjacent points (see Fig. 3 and [14]). In this paper, we refer to quasicrystals with three (respectively two) possible distances simply as *three-tile sequences* (respectively *two-tile sequences*).

As they are periodic, the sequences combined in the APRNG have some lattice structure. The combination design uses the aperiodicity property of the quasicrystals to eliminate these lattice structures. Since the space complexity of the quasicrystal generation is logarithmic, only a finite segment of the theoretical sequence can be generated which length depends only on the memory size (not on the generator parameters).

In the proposed APRNG design, the aperiodic tile sequence generated by a three-tile (respectively two-tile) sequence is used to combine three (respectively two) periodic number sequences. When the periodic sequences are pseudorandom sequences, the resulting sequence will be a pseudorandom sequence.

The paper is organized as follows. In the first section we recall the design of APRNGs. In Section 2, we consider the class of APRNGs combining two PRNGs using two-tile sequences. We discuss the parallel generation of such APRNGs and show that a sufficiently long (finite) segment of the sequence generated by any of these APRNGs has no lattice structure in $t$-dimensional space. In Section 3 we briefly describe the statistical tests we will consider and also give some facts about quasicrystals. We present the results of empirical testing of a specific type of APRNGs (LCGs with modulus $2^{31} - 1$ combined using quasicrystals based on $\sqrt{5}$) using the DIEHARD package and the Maurer test. We show the dependence on the type of quasicrystal used (two- or three-tiles) and the chosen PRNGs. The implementation of APRNGs is discussed in Section 4. Since PRNG implementation is thoroughly discussed in the literature, we only look at the quasicrystal points generation. An extended description of the numerical implementation is given along with an overview of the symbolic generation (symbolic generation is discussed in detail in [18]).

## 1. Preliminaries

In this section we give an introduction to quasicrystals and recall the APRNGs design introduced in [6]. In the introduction we state many known properties of quasicrystals that will be used in the proofs of the various theoretical results and in the implementation discussion. We also define the *tile sequence* associated to a quasicrystal and which, in the proposed APRNG design, is used to combine two or three PRNGs.

### 1.1. Quasicrystals

Cut and project quasicrystals (here simply referred to as quasicrystals) may be described geometrically as projections of points from $m$-dimensional *lattices* into $n$-dimensional spaces, where $m > n$. The quasicrystals discussed in this paper are projections from a two-dimensional lattice $\mathbb{Z}^2$ to a straight line with a specific irrational slope. We consider lines whose slope $\beta$ is the greater solution of $x^2 = mx \pm 1$, for some positive integer $m$, and such that $|\beta'| < 1 < \beta$, where $\beta'$ is the second (smallest) root of the equation. There are infinitely many such equations and thus infinitely many such lines. In the empirical statistical tests we only consider lines whose slope is the *golden mean* $\tau = \frac{1}{2}(1+\sqrt{5}) \approx 1.618$, the greater root of $x^2 = x + 1$. The second root, called the *algebraic conjugate* of $\tau$, is $\tau' = \frac{1}{2}(1-\sqrt{5}) \approx -0.618$.

Fig. 1 illustrates the cut and project scheme. It displays points of the two-dimensional lattice $\mathbb{Z}^2$ and a straight line $p_\parallel$ with irrational slope $\beta$. The interval $\Omega = [c, d)$ on the line $p_\perp$ (this second line has slope $\beta'$ and is denoted $p_\perp$ as it is perpendicular to $p_\parallel$ in cases where $x^2 = mx + 1$) determines a strip parallel to the line $p_\parallel$. The quasicrystal is formed by the projections of the lattice points in the strip to the line $p_\parallel$, it is denoted $\Sigma(\Omega)$.

This geometric construction can be described algebraically. To ease the notation, we only give the description for the case $\beta = \tau$ (the general algebraic definition is treated in [12] and [14], and in [5]). We use the following notations.

The quadratic extension $\mathbb{Q}[\tau]$ of the field of rational numbers $\mathbb{Q}$ is defined as the smallest number field containing the rational numbers and $\tau$:

$$\mathbb{Q}[\tau] = \{a + b\tau \mid a, b \in \mathbb{Q}\}.$$

The relation between $\tau$ and $\tau'$ is used to define the following *Galois automorphism* on $\mathbb{Q}[\tau]$:

$$x' = (a + b\tau)' = a + b\tau', \quad \forall x \in \mathbb{Q}[\tau].$$

The number $x'$ is called the *conjugate* of $x$. The fact that the mapping $' : \mathbb{Q}[\tau] \to \mathbb{Q}[\tau]$ is an automorphism means that $x' \in \mathbb{Q}[\tau]$, $(x + y)' = x' + y'$, and $(xy)' = x'y'$. The automorphism is of second order, i.e., $x'' = x$.

We will work with the ring of integers

$$\mathbb{Z}[\tau] := \{a + b\tau \mid a, b \in \mathbb{Z}\}.$$

**Definition 1.1.** A *one-dimensional cut and project quasicrystal* (here simply refered to as *quasicrystals*) is the set

$$\Sigma(\Omega) := \left\{x \in \mathbb{Z}[\tau] \mid x' \in \Omega\right\},$$

where $\Omega$ is a bounded interval in $\mathbb{R}$ with non-empty interior. $\Omega$ is called the *acceptance window* of the quasicrystal $\Sigma(\Omega)$.

The quasicrystal points can be ordered on the real line, i.e., the quasicrystal can be written as an increasing sequence $(x_i)_{i\in\mathbb{Z}}$. The sequence of distances between two adjacent quasicrystal points can be written as $(x_{i+1} - x_i)_{i\in\mathbb{Z}}$. There are at most three possible distances between any two adjacent points of a quasicrystal (see Proposition 1.4). We call these distances *tiles*. Indeed, the points of a quasicrystal generate a *tiling* of the real line whose tile types correspond to the possible distances. Although every quasicrystal tiling consists of tiles of at most three types, no quasicrystal tiling is periodic: they are all *aperiodic*.

Quasicrystals have the following important properties.

**Proposition 1.2** [14]. *Let $\Sigma(\Omega)$ be a quasicrystal. $\Sigma(\Omega)$ has the following properties*:

(1) Shifting property: $\Sigma(\Omega + \lambda') = \Sigma(\Omega) + \lambda$, *for any* $\lambda \in \mathbb{Z}[\tau]$.
(2) Scaling property: $\Sigma(\tau^i\Omega) = (\tau')^i\Sigma(\Omega)$, *for any* $i \in \mathbb{Z}$.

These properties show that infinitely many quasicrystals cannot be geometrically distinguished without any *a priori* size ($\tau^i$) and position ($\lambda$) information.

**Corollary 1.3.** *Let $\Sigma(\Omega)$ be a quasicrystal. Up to shifting and scaling, we can assume $0 \in \Omega = [c, d)$ and $d - c \in [1, \tau)$. We will further assume that the starting point (or seed-point) of the quasicrystal segment is $x_0 = 0$.*

Let $\Omega$ be a bounded interval, e.g., $[c,d)$. To ease the notation, we write $\Sigma[c,d)$ rather than $\Sigma([c,d))$.

**Proposition 1.4** [12,14]. *Let $\Sigma(\Omega)$ be a quasicrystal with acceptance window $\Omega=[c,d)$, where $d-c\in[1,\tau)$. Let $(x_i)_{i\in\mathbb{Z}}$ be the ordered sequence of points of $\Sigma(\Omega)$. Then $\Sigma(\Omega)$ is of one of the following two types*:

- *If $d-c=1$, then $\Sigma(\Omega)$ is a* two-tile quasicrystal: *For all $i$, $x_{i+1}-x_i\in\{\tau,\tau^2\}$. Each of these two distances appears infinitely many times in $\Sigma(\Omega)$.*
- *If $d-c\neq 1$, then $\Sigma(\Omega)$ is a* three-tile quasicrystal: *For all $i$, $x_{i+1}-x_i\in\{1,\tau,\tau^2\}$. Each of these three distances appears infinitely many times in $\Sigma(\Omega)$.*

The three distances $1$, $\tau$, and $\tau^2$ are called *short*, *middle*, and *long* (or simply $S$, $M$, and $L$), respectively (note that $\tau^2=1+\tau$, in other words $L=M+S$). If the distance between two adjacent points in a quasicrystal is $x_{i+1}-x_i=\tau^j$, then the distance between their conjugates is $x'_{i+1}-x'_i=(x_{i+1}-x_i)'=(\tau^j)'=(\tau')^j$. Hence the distances $1$, $\tau$, and $\tau^2$ in the quasicrystal are mapped to $1$, $\tau'=-1/\tau$, and $\tau'^2=1/\tau^2$ in the acceptance window.

The point set $\Sigma(\Omega)$ may be splitted into three subsets according to the type of the right adjacent tile. These subsets are then included in three disjoint intervals which partition $\Omega$, as stated in the following lemma.

**Lemma 1.5.** *Let $\Sigma(\Omega)=\Sigma[c,d)$, $d-c\in[1,\tau)$ be a quasicrystal. Let $\Sigma_S$, $\Sigma_M$, and $\Sigma_L$ be sets of points of $\Sigma(\Omega)$ such that $\Sigma_S=\{x_i\in\Sigma(\Omega)\mid x_{i+1}-x_i=1\}$, $\Sigma_M=\{x_i\in\Sigma(\Omega)\mid x_{i+1}-x_i=\tau\}$, and $\Sigma_L=\{x_i\in\Sigma(\Omega)\mid x_{i+1}-x_i=\tau^2\}$. Then the following three equalities hold*:

$$\begin{aligned}\Sigma_S &= \bigl\{x\in\mathbb{Z}[\tau]\mid x'\in\Omega_S=[c,d-1)\bigr\},\\ \Sigma_M &= \left\{x\in\mathbb{Z}[\tau]\mid x'\in\Omega_M=\left[c+\frac{1}{\tau},d\right)\right\},\\ \Sigma_L &= \left\{x\in\mathbb{Z}[\tau]\mid x'\in\Omega_L=\left[d-1,c+\frac{1}{\tau}\right)\right\}.\end{aligned}$$

*In particular, $\Sigma_S$ is empty if $d-c=1$.*

The result of this lemma will be used later in the numerical generation of quasicrystals (Section 4.1). Indeed, knowing to which of the three subintervals ($\Omega_S$, $\Omega_M$, or $\Omega_L$) $x'_i$ belongs, we can easily determine the adjacent tile type and thus generate $x_{i+1}$, i.e., define a function which *steps* from $x_i$ to $x_{i+1}$. This function is referred to as the *stepping function*.

**Definition 1.6.** Let $\Sigma(\Omega)=\Sigma[c,d)$ with $d-c\in[1,\tau)$. We define the *stepping function* $f:\Omega\to\Omega$ as

$$f(x):=\begin{cases}x+1 & \text{for } x\in\Omega_S=[c,d-1),\\ x-\frac{1}{\tau} & \text{for } x\in\Omega_M=\bigl[c+\frac{1}{\tau},d\bigr),\\ x+\frac{1}{\tau^2} & \text{for } x\in\Omega_L=\bigl[d-1,c+\frac{1}{\tau}\bigr).\end{cases}\qquad(1)$$

In other words, if $x_i\in\Sigma(\Omega)$ then $x'_{i+1}=f(x'_i)$, or more generally $x'_{i+k}=f^{(k)}(x'_i)$, where $f^{(k)}$ is the $k$th iteration of the function $f$. This observation easily follows from Lemma 1.5 and Definition 1.6.

Finally, it is known that the density of points in any quasicrystal depends on the size of the acceptance window $\Omega$ and that it is independent of its position on the real axis (the density of points is given as the limit of the number of quasicrystal points in an interval $I$ over the length of $I$ when $I$ tends to $\mathbb{R}$). Moreover $\Sigma_S$, $\Sigma_M$, and $\Sigma_L$ are in fact three quasicrystals with three disjoint acceptance windows $\Omega_S$, $\Omega_M$, and $\Omega_L$. Hence the density of $\Sigma(\Omega)$ is the sum of the densities of $\Sigma_S$, $\Sigma_M$, and $\Sigma_L$.

Quasicrystals may be viewed as infinite words in the letters $S$, $M$ and $L$. The relative frequencies (densities) of the respective letters in these infinite words are functions of the lengths of the subintervals $\Omega_S$ , $\Omega_M$, and $\Omega_L$ as stated in the following lemma.

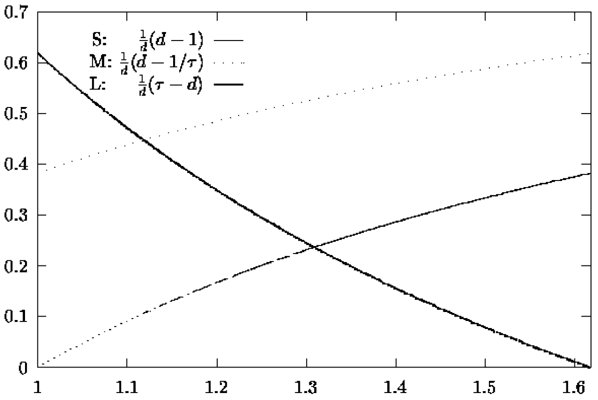
**Lemma 1.7.** *Let $\Sigma[c, c+d)$ be a quasicrystal such that $1 \leqslant d < \tau$. The densities of tiles $S$, $M$, and $L$ are $\frac{1}{d}(d-1)$, $\frac{1}{d}(d-1/\tau)$, and $\frac{1}{d}(\tau-d)$, respectively.*

Fig. 2(a) illustrates the three densities as functions of $d$. Notice how the density of $S$ (respectively $L$) tends to 0 as $d$ decreases to 1 (respectively increases to $\tau$). The densities of $S$ and $L$ are equal when $d-1=\tau-d$, i.e., for $d=\frac{\tau^2}{2}$. The densities of $M$ and $L$ are equal when $d-1/\tau=\tau-d$, i.e., $d=\frac{\tau+2}{2\tau}$.

## 1.2. The aperiodic pseudorandom number generator design

APRNGs were introduced in [6]. Here, we present the design for a three-tile sequence combining any three sequences. An illustration of the design for a two-tile sequence combining any two sequences is given in Fig. 3.

The generator uses one aperiodic three-tile sequence denoted $(t_n)_{n\in\mathbb{N}}$ (generated from $\Sigma(\Omega)$) and three (possibly identical) pseudorandom sequences denoted $(v_n^{(i)})_{n\in\mathbb{N}}$ ($i = 1, 2, 3$). From these four sequences we create a pseudorandom sequence denoted $(z_n)_{n\in\mathbb{N}}$ in the following way. Assume that among



(a)

```
Step $n+1$:
  1. Generate $t_{n+1}$.
  2. if ($t_{n+1} = S$) then
         $z_{n+1} := v^{(1)}_{s+1}$;
         $s := s+1$;
     else if ($t_{n+1} = L$) then
         $z_{n+1} := v^{(3)}_{\ell+1}$;
         $\ell := \ell+1$;
     else $z_{n+1} := v^{(2)}_{m+1}$;
         $m := m+1$
```

(b)

Fig. 2. (a) Tile densities of $\Sigma[c, c+d)$; (b) Algorithm 1: Aperiodic generator.

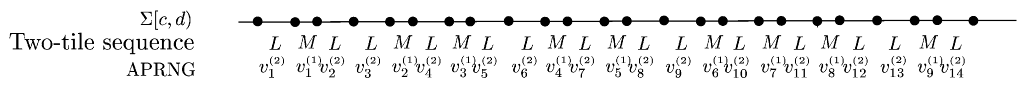

$\Sigma[c,d)$
Two-tile sequence
APRNG
L M L L M L M L L M L M L L M L M L M L L M L
$v_1^{(2)}$ $v_1^{(1)}v_2^{(2)}$ $v_3^{(2)}$ $v_2^{(1)}v_4^{(2)}$ $v_3^{(1)}v_5^{(2)}$ $v_6^{(2)}$ $v_4^{(1)}v_7^{(2)}$ $v_5^{(1)}v_8^{(2)}$ $v_9^{(2)}$ $v_6^{(1)}v_{10}^{(2)}$ $v_7^{(1)}v_{11}^{(2)}$ $v_8^{(1)}v_{12}^{(2)}$ $v_{13}^{(2)}$ $v_9^{(1)}v_{14}^{(2)}$

Fig. 3. The structure of an APRNG.

$t_1, t_2, \dots, t_n$ there are $s$ letters $S$, $m$ letters $M$ and $\ell$ letters $L$ with $s + m + \ell = n$. The $(n+1)$th-step of the algorithm generates $z_{n+1}$ as described in Algorithm 1.

Fig. 3 illustrates a sequence generated by an APRNG using a two-tile sequence.

## 2. Statistical properties

In this section we prove statistical properties of the APRNGs which use a two-tile sequence to combine two arbitrary PRNGs. It is shown in [5] that the sequences produced by such APRNGs have no periodic subsets. Here we show a stronger result: the nonexistence of lattice structure (in any dimension) in the aperiodic pseudorandom sequence.

**Theorem 2.1.** *Let $Z$ be a pseudorandom sequence generated by an APRNG using a two-tile sequence to combine two periodic PRNGs. If the two PRNGs have the same set $\Gamma$ of* (*rational*) *output values, then any sufficiently long finite segment of $Z$ has no lattice structure.* (*In other words, any family of hyperplanes covering all* (*overlapping or nonoverlapping*) *$t$-tuples in a sufficiently long segment of $Z$ also covers all $t$-tuples $(a_1, a_2, \dots, a_t)$ with $a_i \in \Gamma$.*)

(Note: As one would expect, the condition on the minimal length of the sequence to be considered is finite but depends on the dimension size $t$ and if we are considering overlapping or nonoverlapping $t$-tuples. The minimal length to consider is not estimated in the paper.)

The proof of Theorem 2.1 relies on the following proposition (which is proven below in Section 2.1).

**Proposition 2.2.** *Let $Z$ be a pseudorandom sequence generated by an APRNG using a two-tile sequence to combine two periodic PRNGs having the same set $\Gamma$ of* (*rational*) *output values, and let*

$$\nu = \max\{q \in \mathbb{Q} \mid \gamma_1 - \gamma_2 \in q\mathbb{N} \textit{ for all } \gamma_1, \gamma_2 \in \Gamma\}.$$

*If a normal to a family of hyperplanes covering all $t$-tuples in $Z$ is not orthogonal to a vector $E$* (*from the canonical basis*)*, then the distance along $E$ between adjacent hyperplanes in the family divides $\nu$.*

(Note: $\nu$ is the biggest real number such that given $\gamma_1 \in \Gamma$, then for any $\gamma \in \Gamma$ there exists an integer $a$ such that $\gamma = \gamma_1 + a\nu$.)

**Proof of Theorem 2.1.** Let $N$ be a normal vector to a family of hyperplanes covering all $t$-tuples in $Z$. Let $S = \{E_{j_1}, E_{j_2}, \dots, E_{j_k}\}$ be the vectors from the canonical basis not orthogonal to $N$ and let $d_s$ be the distance along $E_{j_s}$ between adjacent hyperplanes of the family. (We denote by $E_j$ the vector $(\delta_{j1}, \delta_{j2}, \dots, \delta_{jt})$.)

Consider the $t$-tuple $P = (z_1, z_2, \dots, z_t)$ (where by construction $z_i \in \Gamma$). (Note that since $P$ is covered by an hyperplane in the family and since $E_j$ is orthogonal to $N$ for all $E_j \notin S$, all $t$-tuples $Y = (y_1, y_2, \dots, y_t)$ with $y_i \in \mathbb{R}$ for $i \neq j_1, j_2, \dots, j_k$ and $y_{j_i} = z_{j_i}$ are also covered by an hyperplane in the family, see Fig. 4(a).) For any $\gamma \in \Gamma$ we have that $\Gamma \subset \{\gamma + n \cdot \nu \mid n \in \mathbb{N}\}$ (by definition of $\nu$). Hence the $t$-tuples $(y_1, y_2, \dots, y_t)$ such that for $s = 1, 2, \dots, k$, $y_{j_s} = z_{j_s} + ad_s$ for any $a \in \mathbb{N}$ are covered by the family of hyperplanes (by definition of $d_s$, see Fig. 4).

By Proposition 2.2, $d_s$ divides $\nu$ for all $s = 1, 2, \dots, k$, hence all $t$-tuples $(y_1, y_2, \dots, y_t)$ such that, for $s = 1, 2, \dots, k$, $y_{j_s} = z_{j_s} + a\nu$ for some $a \in \mathbb{N}$ are also covered by the family of hyperplanes. As there are

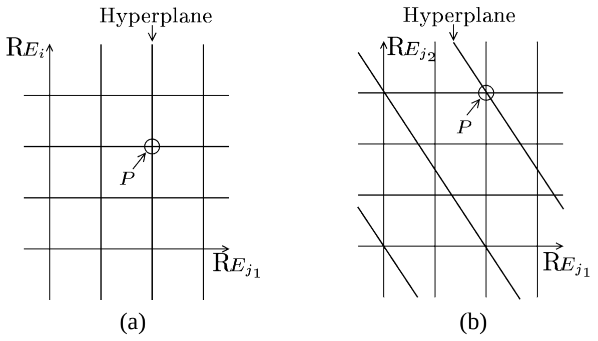


Fig. 4. Hyperplanes trace on selected planes. (a) Case $k < t$. (b) Case $k \geqslant 2$.

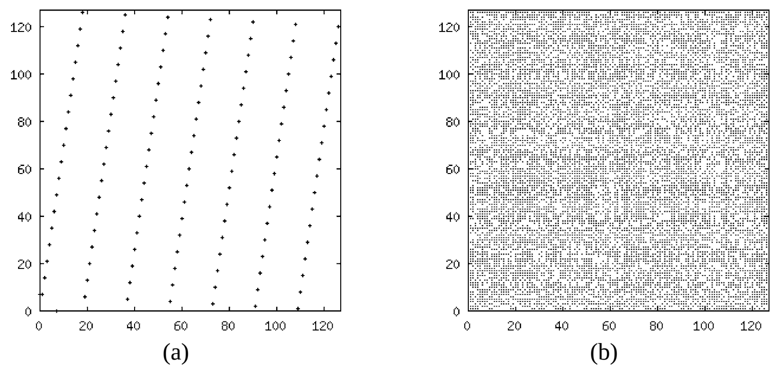


Fig. 5. Example of 2-dimensional structure: (a) Lattice structure (overlapping pairs) of the MCG defined by the recursion $x_{n+1} = 7x_n \pmod{127}$. (Dots are enlarged to be visible.) (b) Overlapping pairs appearing in the $126^2$ first points of the aperiodic sequence generated using a two-tile sequence and the same MCGs.

only finitely many $t$-tuples $(a_1, a_2, \ldots, a_t)$ with $a_i \in \Gamma$, these are covered by the family of hyperplanes if the family also covers the $t$-tuples appearing in a sufficiently long segment of $Z$. □

We illustrate the theorem by giving an example of the 2-dimensional lattice structure of one MCG and of several finite sequences generated by one APRNG combining two copies of this single MCG (using one two-tile sequence). We use the MCG defined by the recursion $x_{n+1} = 7x_n \pmod{127}$ whose 2-dimensional lattice structure is given in Fig. 5(a). Fig. 5(b) illustrates the first $126^2$ points generated by the APRNG. To illustrate the nonuniform distribution of pairs $(x_i, x_{i+1})$ of successive points generated by the APRNG, let us mention that the first $3 \times 126^2$ pairs cover all but three points of the $126^2$ lattice points; the first $4 \times 126^2$ pairs cover all but one point of the lattice; and finally that the first $5 \times 126^2$ pairs cover all the lattice points.

It should be noted that even though examples tend to show that Theorem 2.1 extends to cases of APRNG using three-tile sequences, the proof of Proposition 2.2 presented in this paper does not. Indeed, the proof of Proposition 2.2 relies on four properties (see below) one of which (Property P4) is not true for three-tile sequences. (The subword complexity of any three-tile sequences is between $n + 1$ and $2n + 1$, see [7].)

## *2.1. Proof of Proposition 2.2*

The proof of Proposition 2.2 is based on the following lemma.

**Lemma 2.3.** *Let $Z$ be a pseudorandom sequence generated by an APRNG using a two-tile sequence to combined two periodic PRNGs having sets of* (*rational*) *output values $\Gamma_1$ and $\Gamma_2$, respectively. For any $a \in \Gamma_1$, $b \in \Gamma_2$ and positive integers $\ell$ and $t$, there exist infinitely many integers $m$ and $p$ such that $(z_m, z_{m+1}, \dots, z_{m+\ell-1}) = (z_{m+pt}, z_{m+pt+1}, \dots, z_{m+pt+\ell-1})$, $z_{m+\ell} = a$ and $z_{m+pt+\ell} = b$.*

**Proof.** To ease the notation, we only give the proof for the special case where the two-tile sequence is based on $\tau$, the golden ratio. The proof of the general case follows similarly.

Let $(x_n)_{n\in\mathbb{Z}}$ be the quasicrystal points, $(t_n)_{n\in\mathbb{Z}}$ be the associated tile sequence and $Z = (z_n)_{n\in\mathbb{N}}$. (Again, to ease the notation we assume $x_0 = 0$.) Let $\alpha(\ell)$ be the number of tiles of type $L$ in $(t_n)_{n=0}^{\ell-1}$ (then $\beta(\ell) = \ell - \alpha(\ell)$ is the number of tiles of type $M$ in $(t_n)_{n=0}^{\ell-1}$).

Without loss of generality, we can assume that the two PRNGs have the same period $\mathcal{P}$: in cases where the two PRNGs have distinct periods $P_1$ and $P_2$, the result follows by considering $\mathcal{P} = \operatorname{lcm}(P_1, P_2)$. Moreover we can assume tiles of type $L$ are associated to numbers in $\Gamma_1$ and tiles of type $M$ are associated to numbers in $\Gamma_2$.

The proof is based on the following four properties. Let $a \in \Gamma_1$ and $b \in \Gamma_2$.

(P1) Since the PRNGs are periodic of period $\mathcal{P}$, the pseudorandom number associated to the $i$th tile $M$ (mod $\mathcal{P}$) is fixed, and similarly for $L$ (see Fig. 3). More precisely there exists an integer $i_1$ such that if $t_n = L$ and $\alpha(n) = i_1 \mod \mathcal{P}$ then $z_n = a$; similarly there exists an integer $i_2$ such that if $t_n = M$ and $\beta(n) = i_2 \mod \mathcal{P}$ then $z_n = b$.

(P2) Since $\mathbb{Z}[\tau]$ is dense in $\mathbb{R}$, the sets

$$\Delta = \left\{ \frac{A}{\tau^2} - \frac{B}{\tau} \;\middle|\; A, B \in \mathbb{N} \right\} \quad \text{and} \quad \Theta = \left\{ \frac{A}{\tau^2} - \frac{B}{\tau} \;\middle|\; A, B \in \mathbb{Z} \right\},$$

are dense in $\mathbb{R}$. Note that the sets $\Delta' = \{x' \mid x \in \Delta\}$ and $\Theta' = \{x' \mid x \in \Theta\}$ have no accumulating point in $\mathbb{R}$.

(P3) Let $y' \in \mathbb{Z}[\tau] \cap [c, c+1)$. By definition $y$ is a quasicrystal point, say $y = x_n$. Since $1/\tau^2$ and $1/\tau$ form an integer basis of $\mathbb{Z}[\tau]$, there exist two integers $A$ and $B$ such that $y' = A/\tau^2 - B/\tau$. Moreover we assume $x_0 = 0 \in [c, c+1)$, therefore the integer coefficients $A$ and $B$ are either both non-negative (when $n > 0$) or both non-positive (when $n \leqslant 0$). In any case $y = x_n = x_{A+B}$.

(P4) The subword complexity of a two-tile sequence is $n+1$ (the number of different subwords of length $n$ in a two-tile sequence is $n+1$). Hence, for any positive integer $n$ there exist two intervals $I_n, J_n \subset \Omega$ and an $n$-letter word $w_n$ such that

$$x'_i \in I_n \iff t_i t_{i+1} \cdots t_{i+n-1} t_{i+n} = w_n L,$$
$$x'_j \in J_n \iff t_j t_{j+1} \cdots t_{j+n-1} t_{j+n} = w_n M.$$

Let $\ell$ and $t$ be any positive integers and let $I_\ell$ and $J_\ell$ be the intervals given by Property P4 and $w_\ell$ be the associated $\ell$-letter word. Moreover let $k_1$ be the number of tiles $L$ in $w_\ell$ and $k_2 = \ell - k_1$ the number of tiles $M$ in $w_\ell$.

To prove Lemma 2.3 it is sufficient, from Property P2, to find infinitely many integers $m$ and $q = pt$ such that $x'_m \in I_\ell$ and $x'_{m+pt} \in J_\ell$ with $\alpha(m) = \alpha(m + pt) = i_1 - k_1 \bmod \mathcal{P}$ and $\beta(m) = \beta(m + pt) = i_2 - k_2 \bmod \mathcal{P}$. Indeed, then the tile sequences $t_m t_{m+1} \cdots t_{m+\ell-1}$ and $t_{m+pt} t_{m+pt+1} \cdots t_{m+pt+\ell-1}$ are identical and so are the pseudorandom sequences $z_m z_{m+1} \cdots z_{m+\ell-1}$ and $z_{m+pt} z_{m+pt+1} \cdots z_{m+pt+\ell-1}$; $z_{m+\ell} = a$ and $z_{m+pt+\ell} = b$ follows from P1. We now show that infinitely many such $m$ and $q$ exist.

From Property P2, there exist infinitely many pairs $(A, B)$ of positive integers such that

$$y_1 := \frac{i_1 - k_1}{\tau^2} - \frac{i_2 - k_2}{\tau} + \left(A\frac{1}{\tau^2} - B\frac{1}{\tau}\right)\mathcal{P} \in I_\ell.$$

It follows from Property P3 that for each such $y_1$ there exists a positive integer $m$ such that $y_1 = x'_m$ with $\alpha(m) = i_1 - k_1 \bmod \mathcal{P}$ and $\beta(m) = i_2 - k_2 \bmod \mathcal{P}$, thus from Property P1 we have that $z_{m+\ell} = a$ as $t_{m+\ell} = L$ and $\alpha(m + \ell) = i_1 \bmod \mathcal{P}$.

Finally, from Property P2, there exist infinitely many pairs $(A, B)$ of positive integers such that

$$y_2 := x'_m + \left(A\frac{1}{\tau^2} - B\frac{1}{\tau}\right)t \cdot \mathcal{P} \in J(w_\ell),$$

and from Property P3, for each such $y_2$ we have that $y_2 = x'_{m+pt}$ with $p = (A + B)\mathcal{P}$. Hence

$$\alpha(m + pt) = i_1 - k_1 \bmod \mathcal{P} \quad \text{and} \quad \beta(m + pt) = i_2 - k_2 \bmod \mathcal{P},$$

thus

$$\alpha(m + pt + k) = \alpha(m + k) \bmod \mathcal{P} \quad \text{and} \quad \beta(m + pt + k) = \beta(m + k) \bmod \mathcal{P}$$

for $k = 0, 1, \ldots, \ell - 1$, and $z_{m+pt+\ell} = b$ from Property P1. □

**Proof of Proposition 2.2.** Let $(HP^t_n)_{n\in\mathbb{Z}}$ be a family of hyperplanes covering all $t$-tuples in $Z$ and having the vector $N$ as a normal, and let $\{E_{i_1}, E_{i_2}, \ldots, E_{i_k}\}$ ($1 \leqslant k \leqslant t$ and $i_j < i_{j+1}$) be the set of vectors from the canonical basis which are not orthogonal to $N$. Denote by $d_j$ the distance along $E_{i_j}$ between adjacent hyperplanes in the family.

The proof is done by induction. Let $i_s$ be the largest index such that $d_s$ does not divide $\nu$, where

$$\nu = \max\{q \in \mathbb{Q} \mid \gamma_1 - \gamma_2 \in q\mathbb{N} \text{ for all } \gamma_1, \gamma_2 \in \Gamma\}.$$

From Lemma 2.3 (with $\Gamma_1 = \Gamma_2 = \Gamma$), we have that for any two $a, b \in \Gamma$, there exist two (in fact infinitely many) $i_s$-tuples $R_1 = (z_{r_1}, z_{r_1+1}, \ldots, z_{r_1+i_s-1})$ and $R_2 = (z_{r_2}, z_{r_2+1}, \ldots, z_{r_2+i_s-1})$ such that $z_{r_1+\ell} = z_{r_2+\ell}$ for $\ell \leqslant i_s - 2$, $z_{r_1+i_s-1} = a$ and $z_{r_2+i_s-1} = b$. (Note: when considering nonoverlapping $t$-tuples in $Z$, $r_2$ has to be of the form $r_2 = r_1 + pt$ for some integer $p$, whereas when considering overlapping $t$-tuples in $Z$, no restriction has to be made on the value of $r_2$.)

Moreover all $t$-tuples $(y_1, y_2, \ldots, y_t)$ with $y_i = z_{r_i}$ for $i \leqslant i_s - 2$, $y_{i_s-1} \in \{\gamma_1, \gamma_2\}$ and $y_i \in \Gamma$ for $i \geqslant i_s - 1$ are covered by $(HP^t_n)_{n\in\mathbb{Z}}$. (This follows directly from the fact that $R_1$ and $R_2$ are covered by $(HP^t_n)_{n\in\mathbb{Z}}$ and, for $i > i_s$, $E_i$ is either orthogonal to $N$ or the distance along $E_i$ between adjacent hyperplanes in $(HP^t_n)_{n\in\mathbb{Z}}$ divides $\nu$.)

Hence there exist two $t$-tuples $(x_1, x_2, \ldots, x_t)$ and $(y_1, y_2, \ldots, y_t)$ with $x_i = y_i$ for $i \neq i_s - 1$, $x_{i_s-1} = a$ and $y_{i_s-1} = b$, and which are covered by $(HP^t_n)_{n\in\mathbb{Z}}$. This implies that $d_s$ divides $\gamma_1 - \gamma_2$: it is the contradiction from which the result follows. □

## 3. Empirical tests: DIEHARD package and Maurer tests

In this section we give empirical testing results we obtained for APRNGs combining pseudorandom sequences generated by linear congruential generators (LCGs ) with modulus $2^{31} - 1$. The tests that were performed are the ones included in the DIEHARD test suite and the Maurer test. We also briefly present the tests.

### *3.1. The DIEHARD package and the Maurer test*

DIEHARD performes a series of "stringent" statistical tests which most popular generators tended to fail. In its present version [11], it is a set of 15 statistical tests the description of which may be found in [11]. DIEHARD is performed on integer-bit representations of the pseudorandom numbers and most of the tests return a *p-value* which should be uniform on $[0, 1)$ if the input file contains truly independent random bits. The C source code of the DIEHARD test suite (and precompiled versions for some platforms) is available at `http://stat.fsu.edu/~geo/diehard.html`.

The Maurer test is bit oriented and made on bit sequences. It was introduced in [15] and designed to measure the cryptographic significance of defects a random bit generator may have. Since it should not be possible to significantly compress the output sequence of a random bit generator, a bit sequence fails the test if it can be significantly compressed (see [16]).[1]

### *3.2. The empirical tests results*

The results of the empirical testing are given in the following tables. These tests are made only for APRNGs with quasicrystals based on $\tau = (1 + \sqrt{5})/2$. The parameters of the various LCGs we will be using in the testing are listed in Table 1.

The properties of the APRNGs depend highly on the properties of the PRNGs (here LCGs for the empirical testing) they use and the way the quasicrystal combines them. Moreover, the combination is controlled by the tile-types distribution. To illustrate these dependencies, we chose LCGs with various

Table 1
Various MCGs used in APRNGs (all with modulus $2^{31} - 1$)

| MCG | Multiplier | Period | Suggested | Grade |
|---|---|---|---|---|
| $\text{MCG}_1$ | 1583458089 | $2^{31} - 2$ | [10, p. 254] | A |
| $\text{MCG}_2$ | $7^5 = 16807$ | $2^{31} - 2$ | [9, p. 106] | A |
| $\text{MCG}_3$ | 62089911 | $2^{31} - 2$ | [9, p. 106] | A |
| $\text{MCG}_4$ | $7^3 = 343$ | $2^{31} - 2$ | | B |
| $\text{MCG}_5$ | $2^8 + 1 = 257$ | $2^{31} - 2$ | | C |
| $\text{MCG}_6$ | $2^8 + 2^2 = 260$ | 23091222 | | D |
| $\text{MCG}_7$ | $2^8 = 256$ | 30 | | F |

[1] The test uses a function $C(L, K)$ which is heuristically approximated in [15]. In [3], the authors compute the precise expression of this function $C(L, K)$ and state that the heuristic estimate may make the Maurer test as much as 2.67 times more permissive. We have carried out the test for both forms of the function and no significant difference was noticed.

statistical properties (see Table 1) and quasicrystals with different tile densities, where these densities are denoted $d_S$, $d_M$ and $d_L$, respectively.

A single two-tile sequence, denoted $T_{d_M<d_L}$, is used to illustrate the APRNGs with two-tile sequences as the tile densities are the same for all two-tile sequences. Three three-tile sequences, denoted $T_{d_S<d_L=d_M}$, $T_{d_S=d_L<d_M}$ and $T_{d_S<d_L<d_M}$, are used to illustrate the APRNGs with three-tile sequences.

In every tested APRNG we use MCGs (LCGs with increment set to 0) rather than general LCGs as it improves slightly the generation speed of the algorithm. The MCG seed point considered is always set to 1. In the last column labeled Grade, we heuristically rank the various MCGs we use. The ranking is made with respect to "how well" they seem to pass (or fail) the empirical tests we make (see Tables 2–7).

We first test the generators using the DIEHARD test suite. We say that a generator fails a given test if at least one $p$-value is greater than 0.9996 or smaller than 0.0004. Otherwise, we say that the generator passes the given test. In the three following tables, we state how many tests the given generators fails (F).

Table 2
Empirical statistical results for MCGs using the DIEHARD test suite

| MCG | F | Grade |
|---|---|---|
| $MCG_1$ | – | A |
| $MCG_2$ | – | A |
| $MCG_3$ | – | A |
| $MCG_4$ | 5 | C |
| $MCG_5$ | 4 | C |
| $MCG_6$ | 8 | D |
| $MCG_7$ | 15 | F |

Table 3
Empirical statistical results using the DIEHARD test suite for APRNGs with two-tile sequence

| | $M$ tile | $L$ tile | F | Grade |
|---|---|---|---|---|
| 1 | $MCG_4$ | $MCG_1$ | – | A |
| 2 | $MCG_1$ | $MCG_4$ | 2 | B |
| 3 | $MCG_5$ | $MCG_1$ | – | A |
| 4 | $MCG_1$ | $MCG_5$ | 3 | C |
| 5 | $MCG_1$ | $MCG_2$ | – | A |
| 6 | $MCG_6$ | $MCG_1$ | 2 | B |
| 7 | $MCG_1$ | $MCG_6$ | 8 | D |
| 8 | $MCG_7$ | $MCG_1$ | 15 | F |

Table 4
Empirical statistical results using the DIEHARD test suite for APRNGs with three-tile sequence

| | Three-tile sequence | $S$ tile | $M$ tile | $L$ tile | F | Grade |
|---|---|---|---|---|---|---|
| 1 | $T_{d_S<d_L=d_M}$ | $MCG_1$ | $MCG_2$ | $MCG_3$ | – | A |
| 2 | $T_{d_S<d_L=d_M}$ | $MCG_1$ | $MCG_2$ | $MCG_4$ | 2 | B |
| 3 | $T_{d_S=d_L<d_M}$ | $MCG_1$ | $MCG_2$ | $MCG_4$ | 1 | B |
| 4 | $T_{d_L<d_S<d_M}$ | $MCG_1$ | $MCG_2$ | $MCG_4$ | 1 | B |
| 5 | $T_{d_S<d_L=d_M}$ | $MCG_1$ | $MCG_2$ | $MCG_6$ | 3 | C |
| 6 | $T_{d_S=d_L<d_M}$ | $MCG_1$ | $MCG_2$ | $MCG_6$ | – | A |
| 7 | $T_{d_L<d_S<d_M}$ | $MCG_1$ | $MCG_2$ | $MCG_6$ | 1 | B |
| 8 | $T_{d_S<d_L=d_M}$ | $MCG_1$ | $MCG_4$ | $MCG_5$ | 1 | B |
| 9 | $T_{d_S=d_L<d_M}$ | $MCG_1$ | $MCG_4$ | $MCG_5$ | 1 | B |
| 10 | $T_{d_L<d_S<d_M}$ | $MCG_1$ | $MCG_4$ | $MCG_5$ | 1 | B |
| 11 | $T_{d_S<d_L<d_M}$ | $MCG_7$ | $MCG_1$ | $MCG_2$ | 15 | F |

The following tables give the results of the Maurer test applied to the generators. We consider the Maurer test for bit blocks of size 6 through 16. We say a test fails if the $N(0,1)$ statistic $Z_\mu$ calculated in the test has significance level $\alpha$ less than 0.01. In cases where the generator is said to pass the test, we list the block length for which $1.2816 < |Z_\mu| < 1.6449$ and for which $1.6449 < |Z_\mu| < 2.5758$, i.e., for which the test has significance level $\alpha$ in $(0.1, 0.2)$ and $(0.01, 0.1)$, respectively.

The empirical testing illustrates some of the dependencies of selected statistics of the aperiodic pseudorandom sequences on the quasicrystal and the LCGs that the aperiodic generators use. The dependency on the quasicrystal reflects the nonuniform distribution of the tiles in the tile sequence. As one intuitively thinks, for a given triple of LCGs, the best statistics will be obtained in cases where the tile with the smallest density (appearing less often) is associated to the LCG of the triple having the worst statistics. In Tables 4 and 7, each group of lines $(2, 3, 4)$, $(5, 6, 7)$ and $(8, 9, 10)$ illustrates this dependency.

The dependency on the LCGs is also obvious: the statistics of the aperiodic pseudorandom sequence will be best when all LCGs have very good properties. In Tables 4 and 7, each group of lines $(1, 2, 5, 8)$, $(3, 6, 9)$ and $(4, 7, 10, 11)$ illustrates the dependency of an aperiodic generator and various triples of LCGs. Tables 3 and 6 illustrate the dependency of an aperiodic generator and various pairs of LCGs.

Table 5
Empirical statistical results for MCGs using the Maurer test

| MCG | $\alpha > 0.2$ | $0.1 < \alpha < 0.2$ | $0.01 < \alpha < 0.1$ | Grade |
|---|---|---|---|---|
| $MCG_1$ | 6–9, 11–16 | – | 10 | C |
| $MCG_2$ | 6–13, 15, 16 | 14 | – | B |
| $MCG_3$ | 7, 8, 11–16 | 9, 10 | 6 | C |
| $MCG_4$ | 6–9, 12–14, 16 | 10, 11 | 15 | B |
| $MCG_5$ | 6–11, 14–16 | 12 | 13 | C |
| $MCG_6$ | 6–8 | – | 9, 10 | D[a] |
| $MCG_7$ | – | – | – | F[b] |

[a] The block lengths 11 to 16 have significance level $\alpha < 0.01$.
[b] All block lengths 6 to 16 have significance level $\alpha < 0.01$.

Table 6
Empirical statistical results using the Maurer test for APRNGs with two-tile sequence

| | $M$ tile | $L$ tile | $\alpha > 0.2$ | $0.1 < \alpha < 0.2$ | $0.01 < \alpha < 0.1$ | Grade |
|---|---|---|---|---|---|---|
| 1 | $MCG_2$ | $MCG_1$ | 6–13, 15, 16 | 14 | – | A |
| 2 | $MCG_4$ | $MCG_1$ | 6–16 | – | – | A |
| 3 | $MCG_1$ | $MCG_4$ | 6, 7, 10–16 | – | 8, 9 | C |
| 4 | $MCG_5$ | $MCG_1$ | 7–10, 12–16 | 6, 11 | – | B |
| 5 | $MCG_1$ | $MCG_5$ | 6–8, 10, 12–16 | 11 | – | D[a] |
| 6 | $MCG_6$ | $MCG_1$ | 6–9, 11–16 | 10 | – | B |
| 7 | $MCG_1$ | $MCG_6$ | 6–9, 12 | – | 10 | D[b] |
| 8 | $MCG_7$ | $MCG_1$ | – | – | – | F[c] |

[a] The block length 9 has significance level $\alpha < 0.01$.
[b] The block lengths 11 and 13 to 16 have significance level $\alpha < 0.01$.
[c] All block lengths 6 to 16 have significance level $\alpha < 0.01$.

Table 7
Empirical statistical results using the Maurer test for APRNGs with a three-tile sequence

| | Three-tile sequence | $S$ tile | $M$ tile | $L$ tile | $\alpha > 0.2$ | $0.1 < \alpha < 0.2$ | $0.01 < \alpha < 0.1$ | Grade |
|---|---|---|---|---|---|---|---|---|
| 1 | $T_{d_S<d_L=d_M}$ | $MCG_1$ | $MCG_2$ | $MCG_3$ | 6–16 | – | – | A |
| 2 | $T_{d_S<d_L=d_M}$ | $MCG_1$ | $MCG_2$ | $MCG_4$ | 8–16 | 6, 10 12, 15 | 7 | C |
| 3 | $T_{d_S=d_L<d_M}$ | $MCG_1$ | $MCG_2$ | $MCG_4$ | 6, 8, 9, 11 13, 15, 16 | 14 | 7, 10, 12 | C |
| 4 | $T_{d_L<d_S<d_M}$ | $MCG_1$ | $MCG_2$ | $MCG_4$ | 6–12, 14, 15 | 13 | 16 | B |
| 5 | $T_{d_S<d_L=d_M}$ | $MCG_1$ | $MCG_2$ | $MCG_6$ | 7, 10–12 14–16 | 6, 13 | 8, 9 | C |
| 6 | $T_{d_S=d_L<d_M}$ | $MCG_1$ | $MCG_2$ | $MCG_6$ | 8–16 | – | 7 | D[a] |
| 7 | $T_{d_L<d_S<d_M}$ | $MCG_1$ | $MCG_2$ | $MCG_6$ | 6, 7, 9–11 14–16 | 12 | 8 | B |
| 8 | $T_{d_S<d_L=d_M}$ | $MCG_1$ | $MCG_4$ | $MCG_5$ | 6–9, 11–13 15 | 10, 14, 16 | – | C |
| 9 | $T_{d_S=d_L<d_M}$ | $MCG_1$ | $MCG_4$ | $MCG_5$ | 7, 9–16 | 6 | 8 | B |
| 10 | $T_{d_L<d_S<d_M}$ | $MCG_1$ | $MCG_4$ | $MCG_5$ | 6, 7, 9–16 | – | 8 | B |
| 11 | $T_{d_S<d_L<d_M}$ | $MCG_7$ | $MCG_1$ | $MCG_2$ | – | – | – | F[b] |

[a] The block length 6 has significance level $\alpha < 0.01$.
[b] All block lengths 6 to 16 have significance level $\alpha < 0.01$.

## 4. Implementation of APRNGs

The implementation of APRNGs can be divided into two independent steps. The first one is the implementation of the PRNGs. In this paper, we only consider implementation with LCGs. The basic idea of implementing LCGs is quite straightforward and has been extensively described in literature, therefore we simply discuss speed optimizations. For example, if 32-bit numbers are used by LCGs, high precision integer types (such as 64-bit `long long` or `__int64`) for multiplication may be needed. These types are not yet supported by all compilers , and if so, they are often emulated by software libraries instead of directly using 64-bit hardware. Such emulations usually imply performance slowdown as they often issue many processor instructions for a single arithmetic operation, therefore it is useful to consider special algorithms for decomposition of multiplication in modulo arithmetics which do not need high precision integer types. Such algorithms have been described for example in [2].

The second step is the quasicrystal generation. A quasicrystal can be generated using two completely different methods: a *numerical* and a *symbolical* methods. The numerical method is relatively easy to implement but has several drawbacks. Computer-stored number types have finite precision, therefore irrational numbers such as $\tau'$ are rounded and the numerical algorithm can occasionally misgenerate a point: it may either generate a point which is not part of the quasicrystal or skip a point of the quasicrystal. Fortunately, these misgenerations are non-accumulating as they simply add or skip a point with no influence on the subsequent generations. The frequency of such misgenerated points can be controlled by choosing the precision.

The symbolic generation is an exact method since it does not use any numbers: it is based on substitutions on finite alphabets. Unlike the numerical method, which has constant space complexity, the space complexity of the symbolic method is a function of $n$, the number of generated tiles. The most efficient known algorithm has space complexity $O(\log n)$.

Let $(x_i)_{i=0}^{\infty}$ be a sequence of quasicrystal points and $(t_i)_{i=0}^{\infty}$ the sequence of tiles (letters) $S$, $M$, and $L$. Let $x_i = a_i + b_i\tau$ denote the $i$th point of the quasicrystal. From Corollary 1.3 (and throughout this section), we need only consider quasicrystals $\Sigma(\Omega)$ with acceptance window length within $[1, \tau)$ and satisfying $0 \in \Omega$. (The tile lengths $S$, $M$ and $L$ are then $1$, $\tau$ and $\tau + 1 = \tau^2$, respectively.) Moreover, we assume the *seed-point* to be $x_0 = 0$.

## 4.1. The numerical method for quasicrystal generation

The numerical method directly generates a quasicrystal point $x_{i+1}$ from its left neighbor $x_i$. It uses the fact that the set of quasicrystal points $\Sigma(\Omega)$ can be split into three subsets $\Sigma_S$, $\Sigma_M$, and $\Sigma_L$ according to the right adjacent tile of the respective points (see Lemma 1.5). Each of these sets consists of elements $x_i$ of $\Sigma(\Omega)$ such that $x_i' \in \Omega_S$, $x_i' \in \Omega_M$, and $x_i' \in \Omega_L$, respectively.

The numerical algorithm is simple. We generate $x_{i+1}$ from $x_i$ using the stepping function $f$ (defined in Definition 1.6) to generate $x_{i+1}'$ from $x_i'$. Indeed, for a given $x_i'$ we determine to which subinterval, $\Omega_S$, $\Omega_M$, or $\Omega_L$, $x_i'$ belongs by finding out the type of the tile adjacent to $x_i$, and then we generate the point $x_{i+1}$. Since the middle tile $\tau$ is more likely than the short tile 1 (Fig. 2(a) and Lemma 1.7), we first check if $x_i'$ belongs to the subinterval $\Omega_M$.[2] We also store the actual value of the integer pair $(a_i, b_i)$ even though they are not needed both by the algorithm or the APRNG. As explained below, these values are used in regular recalculations of $x_i'$ to limit accumulation of rounding errors.

Algorithm 2 shows the algorithm for three-tile quasicrystals. It can be easily simplified for two-tile quasicrystals which only have middle and long tiles: the second **if** and its **then** part are omitted.

### 4.1.1. Floating-point numbers vs. scaled integers

Implementing Algorithm 2 is easy, but the use of irrational numbers will induce precision problems. The precision of the chosen number type is crucial in the comparisons in Step 2 of the algorithm. If, due to a rounded number, any of the comparisons is evaluated improperly, then an inappropriate quasicrystal point and tile are generated. Generally, the higher the precision, the better. On the other hand, for

Algorithm 2: Generating a three-tile quasicrystal $\Sigma[c, d)$

```
1. Let $i := 0$, $a_0 := 0$, and $b_0 := 0$. Therefore $x_0' := 0$.
2. if ($x_i' \geqslant c + \frac{1}{\tau}$) then
       $(a_{i+1}, b_{i+1}) := (a_i, b_i + 1)$; $x_{i+1}' := x_i' + \tau'$; $t_i := M$;
   else if ($x_i' < d - 1$) then
           $(a_{i+1}, b_{i+1}) := (a_i + 1, b_i)$; $x_{i+1}' := x_i' + 1$; $t_i := S$;
        else
           $(a_{i+1}, b_{i+1}) := (a_i + 1, b_i + 1)$; $x_{i+1}' := x_i' + \tau'^2$; $t_i := L$.
3. Let $i := i + 1$.
4. Go to step 2.
```

[2] A computer implementation of this algorithm might have the values of $d - 1$ and $c + 1/\tau$ precalculated and stored in advance so it could simply compare $x_i'$ with these values.

practical portable applications, two APRNGs running on two different machines should generate the same sequence, provided the APRNG parameters are the same. This may not be insured if the machines use different precisions.

The aim is to find a platform-independent way of specifying quasicrystal parameters which can always be given without errors, such as the number of significant digits. In the following paragraphs, we will describe an implementation of the numerical algorithm given above using *scaled integers* instead of floating-point numbers.

The idea is to transform all the real numbers involved in quasicrystal generation into integers. The need of high precision implies that the larger the set of used integers is, the higher the achieved precision is. It follows from Corollary 1.3 that any acceptance window is a subset of the interval $(-\tau, \tau)$, thus all the numbers in Algorithm 2 will be bounded by $-\tau$ and $\tau$. We can therefore scale $\tau$ to the maximum value of the chosen integer type, usually denoted by MAXINT, and $-\tau$ to $-$MAXINT. For example, using an $n$-bit type, $\tau$ could be scaled to $2^{n-1} - 1$ and $-\tau$ to $-2^{n-1} + 1$ (recall that one bit is used for the sign). The scaling factor clearly becomes MAXINT$/\tau$. All the numbers and constants appearing in Algorithm 2 should be multiplied by this constant factor. The acceptance window $[c, d)$ is then given as a pair of integers satisfying:

- $c_{\text{SCALED}} > -\text{MAXINT}$ and $c_{\text{SCALED}} \leqslant 0$,
- $d_{\text{SCALED}} > 0$ and $d_{\text{SCALED}} < \text{MAXINT}$,
- $(d - c)_{\text{SCALED}} \geqslant 1_{\text{SCALED}}$ and $(d - c)_{\text{SCALED}} < \text{MAXINT}$.

Algorithm 3 shows a modified version of Algorithm 2 for the scaled integers. A two-tile version can be again obtained by removing the second **if** and its **then** part. Notice that all the constants are prescaled integers and all variables are also integers.

**Remark 4.1.** If all the needed numerical values are, in the source code, scaled via macros that use directly the scaling factor MAXINT$/\tau$, it may happen that a precompiler rounds some of the values more than necessary. Special attention should be paid to keeping the equation $1_{\text{SCALED}} + \tau'_{\text{SCALED}} = \tau'^2{}_{\text{SCALED}}$. Compilers can round each of the numbers in a different way breaking the equality. Such improper scaling would lead to generation of a sequence differing from the desired quasicrystal.

Algorithm 3: Generating a three-tile quasicrystal using scaled integers

```
1. Let $i := 0$, $a_0 := 0$, and $b_0 := 0$. Therefore $x'_0 = 0$.
2. **if** $(x'_i \geqslant (c + \frac{1}{\tau})_{\text{SCALED}})$ **then**
        $(a_{i+1}, b_{i+1}) := (a_i, b_i + 1)$; $x'_{i+1} := x'_i + \tau'_{\text{SCALED}}$; $t_i := M$;
   **else if** $(x'_i < (d - 1)_{\text{SCALED}})$
        $(a_{i+1}, b_{i+1}) := (a_i + 1, b_i)$; $x'_{i+1} := x'_i + 1_{\text{SCALED}}$; $t_i := S$;
      **else**
        $(a_{i+1}, b_{i+1}) := (a_i + 1, b_i + 1)$; $x'_{i+1} := x'_i + \tau'^2_{\text{SCALED}}$; $t_i := L$.
3. Let $i := i + 1$.
4. Go to step 2.
```

#### *4.1.2. Resynchronization of $x_i'$*

Since in numerical methods (both the floating-point and scaled integers versions) it is impossible to have an exact representation of $\tau'$, the least significant bit of $x_i'$ is always rounded. It is a known fact of elementary numerical mathematics that rounding errors propagate toward more significant bits with every operation. In the worst case, $n$ linear operations influence the least significant $\log_2 n$ bits which then become unusable.

As a consequence, a recalculation or *resynchronization* of $x_i'$ from $(a_i, b_i)$ needs to be done regularly. It eliminates accumulated rounding errors and makes the least significant bits again usable, as explained below. Resynchronization is a time consuming task, as it requires relatively many operations, therefore it may be infeasible to resynchronize after every generated quasicrystal point. The frequency of resynchronization is a compromise between speed and accuracy. It should be performed around Step 3 of Algorithms 2 and 3.

Although we simply need to compute $x' = a + b\tau'$, the high accuracy demands require it to be done very carefully. To illustrate potential problems, we will assume floating-point implementation using the `double` type which has 52 bits of mantissa. For simplicity we also assume that $a$ and $b$ are stored as `double`'s. In such a case we have the following:

- $a$ and $b$ are whole numbers up to 52 bits each.
- Multiplication of two 52-bit numbers $b$ and $\tau'$ theoretically gives a result of up to 104 bits, approximately half of the bits for the integer part and the other half for the fractional part. However, there are only 52 bits of mantissa, therefore only the most significant 52 bits are stored, and the leftover bits are discarded.
- The resulting $x'$ must be within the acceptance window which is a subset of the interval $(-\tau, \tau)$, i.e., its integer part is either $-1$, 0, or 1. In other words, all the bits of the integer part, except of the least significant one, vanish by addition $a + b\tau'$ (recall that $\tau'$ is negative).

In summary, the least significant bits of $b\tau'$ are discarded by the CPU, and the most significant bits are zeroed by adding $a$. The number of the remaining middle bits may be significantly smaller than the precision (52 bits ). In other words, the higher $a$ and $b$ are, the less precise the result is. In extreme cases, when $b$ approaches $2^{52}$, only few bits are kept undamaged. A solution to avoid such situations is to use special arithmetics that does not throw away the least significant 52 bits: scaled $n$-bit integers for example. Multiplication of two integers throws away the upper half of the result as a consequence of the modulo $2^n$ arithmetics. We do not need the upper half of the result of the multiplication $b\tau'$ since it is negative and would be vanished by adding $a$.

Another problem also arises: we only keep a limited number of fractional digits of $\tau'$ (the rounded value). In the scaled integers version, this problem applies also to the scaled value of 1. Consider, for example, the scaled integers method. The $i$th bit of $b\tau'$ can be written as a sum of several bitwise multiplications plus the carryover from the $(i-1)$th bit (in this context, $z_j$ denotes the $j$th bit of $z$, $z_0$ is the least significant bit):

$$(b\tau')_i = \left( \sum_{\substack{\forall j,k \\ j+k=i}} b_j \tau_k' + \text{carryover}_{i-1} \right) \bmod 2. \tag{2}$$

Both indices $j$ and $k$ run through values $\{0,\dots,n-1\}$ (for $n$-bit integers). The accuracy of bits of $b\tau'$ with small $i$ would be higher if we allowed also negative values of the indices $j$ and $k$. Negative indices correspond to the fractional part of the binary expansions of both numbers. Since $b$ is an integer, all its fractional digits are zero, but $\tau'$ has an infinite expansion, and this even when scaled. Therefore we can gain much better precision by adding extra fractional digits to (scaled) $\tau'$.

These considerations lead us to do resynchronization using "double" precision numbers in both scaled integers and floating-point implementations. We describe this process on $n$-bit scaled integers. Both 1 and $\tau'$ must be scaled into $2n$-bit integers. The upper $n$ bits are the same as in the $n$-bit scaling we have considered so far, while the least significant $n$ bits are the added fractional digits. Recall that a multiplication of $2n$-bit and $n$-bit numbers gives a $3n$-bit result. $a$ and $b$ must be multiplied with these "long" integers and only the middle $n$-bit part of the final result is taken. The upper $n$-bit part of the $3n$-bit sum $a+b\tau'$ corresponds to digits of order greater than the order of $\tau$. The middle $n$ bits correspond to the scaled range $(-\tau,\tau)$, while the lowest $n$ bits are the added less significant bits. This issue can be, again, solved very easily using scaled integers, simulating or using (if available) $2n$-bit (such as 64-bit) integers. However, in the floating-point implementation, concatenating two `double`'s in order to get a "double double" is a much harder task.

### *4.1.3. Unwanted features of the numerical algorithm*

Although the effects of roundings are significantly reduced by the double precision resynchronization, they can never be eliminated. Let $x_i'$ be such that its exact unrounded value is in the interval $(d-1-\varepsilon_0, d-1)$, where $\varepsilon_0$ is smaller than the effective precision. It can happen that our numerical generator evaluates (rounded) $x_i'$ as being greater than $d-1$ therefore generating $x_{i+1}' = x_i' + \tau'^2$ while the correct values for $x_{i+1}'$ and $x_{i+2}'$ are $x_i'+1$ and $x_i'+1+\tau' = x_i' + \tau'^2$ respectively. In other words, the generator generates a long tile rather than the expected two-tile sequence of a short tile and a middle tile, i.e., it does not generate quasicrystal point $x_i+1$ but only its right neighbor. Fortunately, the rest of the quasicrystal sequence is undamaged by this misgenerated tile.

Similarly, if the unrounded value of $x_i'$ is in $(d-1, d-1+\varepsilon_0)$, the generator may generate a short tile and a middle tile rather than a single long tile. Only two other types of misgeneration can appear in three-tile quasicrystals, and this if $|x_i' - (c+\frac{1}{\tau})| < \varepsilon_0$. Table 8 lists all 4 possible cases for three-tile quasicrystals. It also lists the two similar cases that can happen in two-tile quasicrystals.

The only consequence of limited precision is that the generator can skip or insert isolated points to a quasicrystal, but such misgenerations do not influence the subsequent generation. The frequency of such misgenerations of course depends on the chosen integer or floating-point type, its precision, and the frequency of resynchronization. Table 9 shows results for a two-tile quasicrystal with acceptance window

Table 8
Possible misgenerations in numerically generated quasicrystals

| Real quasicrystal | Numerical generator | Place of misgeneration | Type of quasicrystal |
|---|---|---|---|
| Long | Middle + Short | $c+1/\tau+\varepsilon_0$ | Three-tile |
| Long | Short + Middle | $d-1-\varepsilon_0$ | Three-tile |
| Middle + Short | Long | $c+1-\varepsilon_0/\tau$ | Three-tile |
| Short + Middle | Long | $d-1+\varepsilon_0$ | Three-tile |
| Middle + Long | Long + Middle | $c+1/\tau-\varepsilon_0$ | Two-tile |
| Long + Middle | Middle + Long | $c+1/\tau+\varepsilon_0$ | Two-tile |

Table 9
Number of misgenerated tiles of $\Sigma[-\frac{1}{\tau^2},\frac{1}{\tau})$. $10^9$ quasicrystal points were generated in each test

| Frequency of | `long` | `long long` | `double` |
|---|---|---|---|
| resynchronization | 31 bits | 63 bits | 52 bits |
| 1 | 2 | 0 | 46 |
| 10 | 6 | 0 | 50 |
| 100 | 18 | 0 | 52 |

$[-\frac{1}{\tau^2},\frac{1}{\tau})$. The fact that the results for `long` are better than the results for more precise `double` is due to "double" precision resynchronization which we have implemented for scaled integers only.

Another problem is that any computer number type can only take a finite number of values, and therefore the generated sequence becomes periodic. Indeed, once all possible values of the number type are generated, $x'_i$ necessarily equals to some $x'_j$ for some $j < i$, and thereafter $x'_{i+k} = x'_{j+k}$ for every positive $k$, provided that resynchronization was done both after $x'_{i-l}$ and $x'_{j-l}$ points for some $l$. This implies that the generated sequence has a period of at most $j - i$ points and the repetition begins by the $i$th point at latest. In the very best case, the periodicity begins starting with $x'_{2^n}$, where $n$ is the number of bits used for the mantissa and the sign or the size of the integer type. The period may be significantly extended by *shifting*.

### 4.1.4. *Shifting of acceptance window*

As explained in the previous paragraph , any generator implementation can only generate a finite segment of a quasicrystal which is then repeated infinitely many times. To significantly extend the length of this segment, we introduce an operation called *shifting*. We say that $\widetilde{\Omega}$ is *shifted* with respect to $\Omega$ if there exists $k \in \mathbb{Z}[\tau]$ such that $\widetilde{\Omega} = \Omega + k$ and $0 \in \widetilde{\Omega}$. The number $N$ of tiles that are generated before shifting occurs is set in the implementation of the generator. $N$ should be smaller than the maximum number of different values of $x'$ that can be achieved for any quasicrystal with the chosen number type.

The shifting is performed as follows. When $N$ quasicrystal points are generated, the generator is restarted with a new acceptance window (either $\Omega$ or $\widetilde{\Omega}$) and a new seed-point 0. The selection of the new acceptance window is controlled by another (two-tile) quasicrystal $\Sigma(\Theta)$ which is generated in parallel. When $N$ points of $\Sigma(\Omega)$ (respectively $\Sigma(\widetilde{\Omega})$) are generated, one tile $t$ of $\Sigma(\Theta)$ is generated. If $t$ is $M$ (respectively $L$), then the next acceptance window to be used is $\Omega$ (respectively $\widetilde{\Omega}$). In sum, shifting means that two different finite segments of two quasicrystals are repeated aperiodically. Since $\Sigma(\Theta)$ is used very rarely, it can be generated even symbolically without constraining memory requirements. If $\Sigma(\Theta)$ is generated exactly the same way as $\Sigma(\Omega)$, then the length of the aperiodic segment becomes $N^2$.

## 4.2. *Symbolic method for quasicrystal generation*

Before describing the method, let us introduce some notation. Let $\mathcal{A}$ be a finite alphabet. Let $\theta$ be a mapping assigning a non-empty finite word $\theta(a)$ in the letters of $\mathcal{A}$ to any letter $a \in \mathcal{A}$. We call $\theta$ a *substitution on* $\mathcal{A}$ and $\theta(a)$ the *substitution word* for $a$, where $a \in \mathcal{A}$. A substitution can be extended to words of arbitrary length $n$ as a concatenation: if $\mu = a_1 \cdots a_n \in \mathcal{A}^n$, then $\theta(\mu) = \theta(a_1)\theta(a_2)\cdots\theta(a_n)$.

A quasicrystal can be viewed as an infinite word $\omega$ in an alphabet $\mathcal{A}$ , e.g., $\mathcal{A} = \{S, M, L\}$. The idea of the symbolic generation is to find a substitution $\theta$ and letters $a \in \mathcal{A}$ such that the sequence $(\theta^j(a))_{j=1}^{\infty}$ is a sequence of prefixes of $\omega$ of increasing lengths, starting at the letter corresponding to the given seed-point.

As shown in [13], for every quasicrystal $\Sigma[c, d)$, with $c, d \in \mathbb{Q}[\tau]$, there exist countably many substitutions (one for each positive integer $s$) each of which has exactly one letter $a$, called the *starting letter,* such that $\lim_{j\to\infty}\theta^j(a)$ produces the infinite tile sequence $(t_i)_{i=0}^{\infty}$ starting at $t_0$. Moreover there exists exactly one letter $b$ such that $\lim_{j\to\infty}\theta^j(b) = (t_i)_{i=-\infty}^{-1}$. The algorithm given in [13] for the construction of the substitution gives both starting letters.

The biggest disadvantage of substitutions is that any generator using them has non-constant space complexity as some of the generated tiles have to be stored for further use in subsequent generations. In terms of space complexity, the best known algorithm (described in [18]) using a stack has space complexity $\frac{1}{s}\log_\tau n$, where $n$ is the number of generated tiles.

With limited memory, only a finite segment of a quasicrystal can be generated. If the generator runs out of memory, it can employ shifting described in Section 4.1.4 using the fact that different acceptance windows induce different substitution rules. Then $N$ (the frequency of shifting) is determined by the stack size and the value of the integer $s$. Since the memory needed is only $O(\log n)$, an extremely long sequence of points can be generated without shifting and with a reasonable amount of memory.

## *4.3. Starting generation from any nth tile*

Any two-tile quasicrystal can be generated starting from any $n$th tile, i.e., the first $n - 1$ tiles can be skipped. Here we describe the algorithm for the numerical case. The symbolic case is described in [18].

Skipping the first $n$ points means finding $a_n$ and $b_n$ ($x_n = a_n + b_n\tau$) without calculating all the pairs $(a_i, b_i)$ with $i < n$. In two-tile quasicrystals, there are only two distances, namely $\tau$ and $\tau^2 = \tau + 1$. Therefore $b_n$ is increased by one on every generated tile and thus $b_n = n$. Since $x'_n = a_n - b_n/\tau$ must be in the acceptance window $[c, c + 1)$, $a_n$ can be calculated as the integer $\lceil c + b_n/\tau \rceil = \lfloor c + b_n/\tau + 1 \rfloor$. However , in the case of scaled integers, $c$ is scaled and generally there can be an implementation-specific need to avoid the floating-point division by $\tau$.

The following algorithm for construction of $a_n$ requires $O(\log n)$ operations. Using the greedy algorithm, $b_n$ can be decomposed into a unique sum of Fibonacci numbers $\sum_{i=1}^{l} k_i F_i$ for some $l$ and $k_i \in \{0, 1\}$ for all $i$ (Fibonacci numbers are defined by formula $F_{i+2} = F_{i+1} + F_i$ with initial conditions $F_0 = 0$ and $F_1 = 1$). Then the approximation $\tilde{a}_n$ is defined as $\sum_{i=1}^{l} k_i F_{i-1}$. It can be shown that $\tilde{a}_n - b_n/\tau \in (-\tau, \tau)$, i.e., is exactly within the range of the scaled integers. There is exactly one $j \in \{-1, 0, 1\}$ such that $\tilde{a}_n + j - b_n/\tau \in [c, c + 1)$, therefore $a_n := \tilde{a}_n + j$.

## *4.4. Comparison of numerical and symbolic methods*

Table 10 lists the main advantages and disadvantages of the three different quasicrystal generator implementations described in this paper: the numerical methods using floating-point (FP) numbers and scaled integers, and the symbolic method.

Table 10
Advantages and disadvantages of different methods of quasicrystal generation

| Generator | Advantages | Disadvantages |
|---|---|---|
| Floating-point | Constant space complexity. | Possibility of misgenerated tiles.<br>Resynchronization hard to implement.<br>Shifting may be needed. |
| Scaled integers | Constant space complexity.<br>Easy resynchronization.<br>Usually faster than FP. | Possibility of misgenerated tiles.<br>Shifting may be needed. |
| Substitution rules | Exact generation.<br>Usually fastest generation. | Non-constant space complexity.<br>Size of memory needed depends on acceptance window. |

Table 11
Benchmarks of generation of $10^8$ points of the quasicrystal $\Sigma[-\frac{\tau}{2}, \frac{1}{2\tau})$. Two numerical and two symbolic methods were used. The values indicate time in seconds

| Hardware<br>CPU Speed | OS | Compiler | 32-bit<br>integers | 64-bit<br>doubles | $s = 2$ | $s = 6$ |
|---|---|---|---|---|---|---|
| Pentium II/450MHz | WinNT | MSVC | 6.17 | 7.25 | 9.28 | 4.62 |
| SGI/180MHz | IRIX | SGI C | 28.99 | 28.20 | 54.80 | 24.07 |
| Alpha/600MHz | UNIX | DEC C | 5.35 | 6.75 | 7.27 | 3.57 |
| UltraSparc IIi/400MHz | Sun OS | gcc | 11.07 | 12.09 | 13.81 | 5.76 |

Table 12
Benchmarks of generation of $10^8$ elements of $\mathrm{MCG}_2$ from Table 1. Arithmetics using 32-bit and 64-bit integer types were used. The values indicate time in seconds

| Hardware<br>CPU Speed | OS | Compiler | 32-bit<br>arithmetics | 64-bit<br>arithmetics |
|---|---|---|---|---|
| Pentium II/450MHz | WinNT | MSVC | 12.17 | 23.88 |
| SGI/180MHz | IRIX | SGI C | 37.98 | 49.84 |
| Alpha/600MHz | UNIX | DEC C | 19.37 | 16.35 |
| UltraSparc IIi/400MHz | Sun OS | gcc | 20.29 | 73.81[a] |

[a] A native Sun C compiler produced a code with running time only 23 seconds as gcc's support of 64-bit hardware does not seem to be efficient.

Table 11 provides basic benchmarks of the three implementations. $10^8$ points of the three-tile quasicrystal $\Sigma[-\frac{\tau}{2}, \frac{1}{2\tau})$ were generated numerically using 32-bit scaled integers and 64-bit floating-point doubles, and generated symbolically using substitution trees. The chosen acceptance window is such that the densities of $M$ and $L$ tiles are equal and higher than the density of $S$'s (see Fig. 2(a)). Two sets of substitution rules were created using two values of the parameter $s$ (2 and 6), the time needed for construction of the rules is not included. The results are the best of those that were achieved using different sets of compiler-level optimizations.

Table 13
Benchmarks of generation of $10^8$ elements of APRNG based on combining three identical copies of MCG$_2$ according to the quasicrystal $\Sigma[-\frac{\tau}{2}, \frac{1}{2\tau})$. The values indicate time in seconds

| Hardware CPU Speed | OS | Compiler | 32-bit arithmetics |
|---|---|---|---|
| Pentium II/450MHz | WinNT | MSVC | 20.03 |
| SGI/180MHz | IRIX | SGI C | 67.12 |
| Alpha/600MHz | UNIX | DEC C | 26.55 |
| UltraSparc IIi/400MHz | Sun OS | gcc | 35.15 |

In Table 12 we provide basic benchmarks of two implementations of MCGs to compare the speed of quasicrystal and PRNG generators. The tested MCG was $x_{i+1} = 16807x_i \mod (2^{31} - 1)$. Since intermediate results of the multiplication can be over $2^{32}$, two algorithms were used. The 4th column of the table displays results for decomposition of 64-bit operations into a sequence of 32-bit operations (as described, for example, in [2]). The fifth column shows results for an implementation using 64-bit integers.

Table 13 shows benchmarks of a three-tile APRNG. The tested APRNG combines 3 identical instances of MCG $x_{i+1} = 16807x_i \mod (2^{31} - 1)$ according to a prescription given by the quasicrystal $\Sigma[-\frac{\tau}{2}, \frac{1}{2\tau})$. 32-bit scaled integers were used to generate the tile sequence and the MCGs were implemented using 32-bit operations as well. The running times are slightly higher (1% to 12%) than the sum of the times of separate tests of the tile sequence and MCG generators (the 4th columns of Tables 11 and 12). The reason for that are some extra operations needed by the combined generator. Also the tables show that APRNG requires about 37% to 77% more running time than the corresponding standalone MCG generator. This increase varies accross different hardware platforms on which the tests were performed. It has to be noted that if more complicated periodic generators were used, such as Linear Feedback Shift Registers (LFSR), which require more CPU time than MCG to generate a single bit or number, the relative time increase would be smaller.

## Conclusions and research avenues

The goal of this work was to show that the aperiodicity of quasicrystals can be used to combine periodic pseudorandom sequences to produce an aperiodic pseudorandom sequence. We have shown how quasicrystals can be used to combine two or three periodic pseudorandom sequences to generate an aperiodic pseudorandom sequence with no lattice structure, and this in any dimension. Moreover, we have shown that the performance hit on the pseudorandom number generation is small with respect to the PRNG generation. For simplicity reasons, we have used LCGs to illustrate the combination technic even though any PRNGs can be used. The theoretical results (Section 2) are independent of the chosen pseudorandom sequences.

As this is the first paper investigating the properties of APRNGs, many interesting questions have arisen. For example, we would like to use the properties of quasicrystals to describe the distribution of $t$-tuples in the aperiodic sequences; we would also like to generalize to the three-tile sequence case the theoretical results obtained for the two-tile sequence case. Also, we have only treated 1-dimensional quasicrystals, but one could be interested in multidimensional pseudorandom sequences. The cut and project quasicrystal model can be generalized to any finite dimension $N$ thus generating $N$-dimensional aperiodic tile sequences. Generalizing the design of APRNGs using these multidimensional sequences generates multidimensional pseudorandom sequences. Even though the generalization of the design is simple, obtaining the properties of these multidimensional tile sequences is a very hard task since virtually nothing is known about the properties of $N$-dimensional cut and project quasicrystals.

## Acknowledgements

The authors thank Edita Pelantová and Zuzana Masáková who suggested many improvements. The authors also thank Pierre L'Écuyer for his stimulating discussions. Jan Patera thanks the Centre de Recherches Mathématiques in Montréal, Québec, where part of this work was done.

## References

[1] E. Bombieri, J.E. Taylor, Quasicrystals, tilings and algebraic number theory: Some preliminary connections, Contemp. Math. 64 (1987) 241–260.
[2] P. Bratley, B.L. Fox, L.E. Schrage, A Guide to Simulation, 2nd Edition, Springer, New York, 1987.
[3] J.-S. Coron, D. Naccache, An accurate evaluation of Maurer's universal test, in: S. Tavares, H. Meijer (Eds.), Selected Areas in Cryptography, SAC Kingston ON, 1998, in: Lecture Notes in Comput. Sci., Vol. 1556, Springer, Berlin, 1999, pp. 57–71.
[4] J. Eichenauer, J. Lehn, A non-linear congruential pseudo random number generator, Statist. Papers 27 (1986) 297–301.
[5] L.-S. Guimond, J. Patera, Proving the deterministic period breaking of linear congruential generators using two tile quasicrystals, Math. Comp. 71 (237) (2002) 319–332.
[6] L.-S. Guimond, J. Patera, J. Patera, Combining random number generators using cut and project sequences, DI-CRM Workshop on Mathematical Physics, Prague, 2000, Czech. J. Phys. 51 (4) (2001) 305–311.
[7] L.-S. Guimond, Z. Masáková, E. Pelantová, Combinatorial properties of infinite words associated with cut-and-project sequences, J. Théorie Nombres de Bordeaux, in press.
[8] P. Hellekalek, Inversive pseudorandom number generators: Concepts, results and links, http://random.mat.sbg.ac.at/generators/wsc95/inversive/inversive.html.
[9] D.E. Knuth, The Art of Computer Programming: Seminumerical Methods, Vol. 2, 3rd Edition, Addison-Wesley, Reading, MA, 1998.
[10] P. L'Écuyer, Tables of linear congruential generators of different sizes and good lattice structure, Math. Comp. 68 (225) (1999) 249–260.
[11] G. Marsaglia, Diehard, http://stat.fsu.edu/pub/diehard.html.
[12] Z. Masáková, J. Patera, E. Pelantová, Minimal distances in quasicrystals, J. Phys. A: Math. Gen. 31 (1998) 1539–1552.
[13] Z. Masáková, J. Patera, E. Pelantová, Substitution rules for aperiodic sequences of cut and project type, J. Phys. A: Math. Gen. 33 (48) (2000) 8867–8886.
[14] Z. Masáková, J. Patera, E. Pelantová, Lattice-like properties of quasicrystal models with quadratic irrationalities, in: H.-D. Doebner, V.K. Dobrev, J.-D. Hennig, W. Luecke (Eds.), Proceedings of Quantum Theory and Symmetries, Goslar, 1999, World Scientific, Singapore, 2000, pp. 499–509.
[15] U.M. Maurer, A universal statistical test for random bit generators, in: A.J. Menezes, S.A. Vanstone (Eds.), Advances in Cryptology—CRYPTO '90, in: Lecture Notes in Comput. Sci., Vol. 537, Springer, Berlin, 1991, pp. 409–420.
[16] A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, Handbook of Applied Cryptography, in: Discrete Mathematics and its Applications, CRC Press, Boca Raton, FL, 1997.

[17] H. Niederreiter, Random Number Generation and Quasi-Monte Carlo Methods, in: SIAM CBMS-NSF Regional Conf. Ser. Appl. Math., Vol. 63, SIAM, Philadelphia, PA, 1992.
[18] J. Patera, Generating the Fibonacci chain in $O(\log n)$ space and $O(n)$ time, Phys. Part. Nuclei 33 (2002) 225–234, Part 7.
[19] H. Sugita, Pseudo-random number generator by means of irrational rotation, Monte Carlo Methods Appl. 1 (1) (1995) 35–57.
[20] R. Zieliński, An aperiodic pseudorandom number generator, J. Comput. Appl. Math. 31 (1) (1990) 205–210.