<table>
<thead>
<tr>
<th>Title</th>
<th>Functional Dependency and Degeneracy: Detailed Analysis of the GAuGE System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authors(s)</td>
<td>Nicolau, Miguel; Auger, Anne; Ryan, Conor</td>
</tr>
<tr>
<td>Publication date</td>
<td>2003-10-30</td>
</tr>
<tr>
<td>Publication information</td>
<td>Liardet, P., Collet, P., Fonlupt, C., Lutton, E. and Schoenauer, M.(eds.). Artificial Evolution (Lecture Notes in Computer science Volume 2936)</td>
</tr>
<tr>
<td>Series</td>
<td>Lecture Notes in Computer Science</td>
</tr>
<tr>
<td>Publisher</td>
<td>Springer</td>
</tr>
<tr>
<td>Item record/more information</td>
<td><a href="http://hdl.handle.net/10197/8281">http://hdl.handle.net/10197/8281</a></td>
</tr>
<tr>
<td>Publisher's statement</td>
<td>The final publication is available at Springer via <a href="http://dx.doi.org/10.1007/978-3-540-24621-3_2">http://dx.doi.org/10.1007/978-3-540-24621-3_2</a>.</td>
</tr>
<tr>
<td>Publisher's version (DOI)</td>
<td>10.1007/978-3-540-24621-3_2</td>
</tr>
</tbody>
</table>
Functional dependency and degeneracy: detailed analysis of the GAuGE system

Miguel Nicolau¹, Anne Auger²,³, and Conor Ryan¹

¹ Department Of Computer Science And Information Systems
University of Limerick, Ireland
{Miguel.Nicolau, Conor.Ryan}@ul.ie
² CERMICS – ENPC
Cité Descartes, 77455 Marne-La-Vallée, France
³ INRIA Rocquencourt, Projet Fractales
BP 105, 78153 LE CHESNAY Cedex, France
auger@cermics.enpc.fr

Abstract. This paper explores the mapping process of the GAuGE system, a recently introduced position-independent genetic algorithm, that encodes both the positions and the values of individuals at the genotypic level. A mathematical formalisation of its mapping process is presented, and is used to characterise the functional dependency feature of the system. An analysis of the effect of degeneracy in this functional dependency is then performed, and a mathematical theorem is given, showing that the introduction of degeneracy reduces the position specification bias of individuals. Experimental results are given, that backup these findings.

1 Introduction

GAuGE [16] (Genetic Algorithms using Grammatical Evolution) is a position independent Genetic Algorithm that encodes both the position and value of each variable of a given problem. It suffers from neither under nor over-specification, due to the use of a genotype-to-phenotype mapping process; that is, a position specification is mapped onto a set of available positions in each phenotype string, ensuring that a fixed-length genotype string generates a single value for each element of the phenotype string.

Two of the main biologically inspired features present in GAuGE are the functional dependency and code degeneracy at the genotypic level. This work is an investigation into both features, and their combined effect on the position specifications of the genotype strings, in the initial generation of individuals. To this end, a mathematical analysis of these features is performed, and a theorem is presented, which shows that an increase of degeneracy leads to a weaker functional dependency across the position specifications in each genotype string. These results are supported by experimental evidence.

This paper is organised as follows. Section 2 gives a brief presentation of the Grammatical Evolution system, and Section 3 presents the GAuGE system and its mapping process. Section 4 presents and analyses the experiments conducted, and Section 5 draws some conclusions and possible future work directions.
2 Grammatical Evolution

Grammatical Evolution (GE) \cite{15,14,13} is an automatic programming system that uses grammars to generate code written in any language. It uses a genetic algorithm to generate variable-length binary strings, which are interpreted as a series of integer values (each value being encoded with 8 bits). These values are then used to choose productions from the rules of a given grammar, to map a start symbol onto a program (the phenotype), consisting solely of terminal symbols extracted from the grammar. This genotype-to-phenotype mapping, based on the analogous process in molecular biology, provides a division between the search space (binary strings) and the solution space (evolved programs) \cite{2}.

Another interesting feature of GE is the functional dependency across the genes on each genotype string. Specifically, the function of a gene is dependent on those genes preceding it, as previous choices of grammar productions will dictate which symbol is to be mapped next, and therefore which rule is to be applied. This creates a dependency across the genotype string, ranging from left to right, and has been termed the “ripple” effect \cite{12}.

Finally, the use of degenerate code plays an important role in GE: by using the mod operator to map an integer to the choices of a grammar rule, neutral mutations \cite{8} can take place, as different integers can choose the same production from a given rule. This means that the genotype can be modified without changing the phenotype, allowing variety at the genotypic level.

3 GAuGE

The GAuGE system is based on most of the same biologically inspired features present in GE, and as GE, it also uses a genotype to phenotype mapping process, thus separating the search space (the genotype space) and the solution space (the phenotype space).

The main principle behind GAuGE is the separate encoding of the position and value of each variable, at the genotypic level. This allows the system to adapt its representation of the problem; experiments conducted previously \cite{10} have shown that this feature allowed GAuGE to evolve its representation, to match the salience hierarchy of a given set of problems.

Previous work has used similar approaches and techniques as the ones employed in GAuGE. Some of Bagley’s \cite{1} computer simulations already used an extended string representation to encode both the position and the value of each allele, and used an inversion operator to affect the ordering of genes. Holland \cite{6} later presented modifications to the schema theorem, to include the approximate effect of the inversion operator. To tackle the problems associated with the combination of the inversion and crossover operators, these were later combined into a single operation, and a series of reordering operators were created \cite{11}.

The so-called messy genetic algorithms applied the principle of separating the gene and locus specifications with considerable success ever since their introduction \cite{4}, and have since been followed by many competent GAs.
Work by Bean [3] with the Random Keys Genetic Algorithm (RKGA) hinted that a tight linkage between genes would result in both a smoother transition between parents and offspring when genetic operators are applied, and an error-free mapping to a sequence of ordinal numbers. More recently, Harik [5] has applied the principles of functional dependency in the Linkage Learning Genetic Algorithm (LLGA), in which a chromosome is expressed as a circular list of genes, with the functionality of a gene being dependent on a chosen interpretation point, and the genes between that point and itself.

3.1 Formal definition of the mapping process

For a given problem composed of \( \ell \) variables, a population \( G = (G_i)_{0 \leq i \leq N-1} \) of \( N \) binary strings, is created, within the genotype space \( \mathcal{G} = \{0, 1\}^L \), where \( L \) will be fixed later on. Through a genotype to phenotype mapping process, these strings will be used to encode a population \( P = (P_i)_{0 \leq i \leq N-1} \), of \( N \) potential solutions, from the phenotypic space \( \mathcal{P} \), which can then be evaluated.

As each string \( G_i \) encodes both the position and the value of each variable of a corresponding \( P_i \) string, the length of \( G_i \) will depend on a chosen position field size \( (pfs) \) and value field size \( (vfs) \), i.e., the number of bits used to encode the position and the value of each variable in \( P_i \). The required length of each string \( G_i \) can therefore be calculated by the formula:

\[
L = (pfs + vfs) \times \ell \tag{1}
\]

The mapping process consists in three steps denoted here by

\[
\Phi : G \xrightarrow{\Phi_1} X \xrightarrow{\Phi_2} R \xrightarrow{\Phi_3} P
\]

The first mapping step \( (\Phi_1) \) consists in interpreting each \( G_i \) as a sequence of \( (position,value) \) specification pairs, using the chosen \( pfs \) and \( vfs \) values. In this way, \( G_i \) can be used to create a string \( X_i \), where:

\[
X_i = (X_i^0, X_i^1) = (X_i^0, X_i^1), (X_i^1, X_i^1), \ldots, (X_i^{l-1}, X_i^{l-1})
\]

with \( X_i^j \) being an integer random variable (encoded with \( pfs \) bits from \( G_i \)) that takes its values from \( \{0, \ldots, 2^{pfs} - 1\} \), and \( X_i^j \) an integer random variable (encoded with \( vfs \) bits from \( G_i \)) that takes its values from \( \{0, \ldots, 2^{vfs} - 1\} \).

The second mapping step \( (\Phi_2) \) then consists in using \( X_i \) to create a string of positions \( R_i = (R_i^0, \ldots, R_i^{l-1}) \), and a string of values \( \tilde{R}_i = (\tilde{R}_i^0, \ldots, \tilde{R}_i^{l-1}) \), where \( R_i^j \in \{0, \ldots, \ell - 1\} \) is the position in the phenotype string \( (P_i) \) that will take the value specified by \( \tilde{R}_i^j \).

The elements of these strings are calculated as follows. Assuming \( R_i^0, \ldots, R_i^{l-1} \) are known, the set \( A_i^j = \{0, \ldots, \ell - 1\} \setminus \{R_i^0, \ldots, R_i^{l-1}\} \) is considered. Then the elements of the set \( A_i^j \) are ordered (from the smaller to the greater) into a list of \( \ell - j \) elements, \( A_i^j \), and

\[
R_i^j = (A_i^j)_{X_i^j \ mod \ (\ell-j)} \tag{2}
\]
where \((A^j_i)_{X^j_i \mod (\ell - j)}\) is the element in \((A^j_i)\) at position \(X^j_i \mod (\ell - j)\). Eq. (2) clearly shows that the random variable \(R^j_i\) depends on the previous random variables \(R^0_i, \ldots, R^{j-1}_i\) and on \(i\).

The string of values can be constructed using the following formula:\(^4\)

\[
\tilde{R}^j_i = \hat{X}^j_i \mod \text{range}
\]

where range is the value range of the variables for the given problem. For a binary problem, for example, range = 2.

Finally, through the third mapping step \((\Phi_3)\), the phenotype string \(P_i\), from the phenotypic space \(\mathcal{P} = \{0, \ldots, \text{range}\}^\ell\) can be calculated by using the formula:

\[
P_i^{R_i} = \tilde{R}^j_i
\]

In other words, through a permutation defined by \(R_i\), the elements of \(\tilde{R}_i\) are placed in their final positions.

### 3.2 Example

As an example of the mapping process employed in GAuGE, consider the (simplified) case where one would wish to solve a binary problem of 4 variables (so range = 2 and \(\ell = 4\)). The first step would be to create a population \(G\) of size \(N\); if the values \(pfs = 6\) bits and \(vfs = 2\) bit were chosen, then, according to Eq. (1), \(L = (6 + 2) \times 4 = 32\), that is, each member of \(G\) is a binary string of 32 bits.

Here is how the mapping process translates each of those strings onto a phenotype string \(P_i\). Consider the following individual:

\[
G_i = 00011001000010100001001100010101
\]

The first step is to create the \(X_i\) string, by using the chosen \(pfs\) and \(vfs\) values:

\[
X_i = ((6, 1), (2, 2), (4, 3), (5, 1))
\]

From this string, the positions string \(R_i\) and the values string \(\tilde{R}_i\) can be created. To create \(R_i\), Eq. (2) is used; for the first element \(R^0_i\), this gives us:

\[
A^0_i = \{0, 1, 2, 3\} \quad R^0_i = (A^0_i)_{6 \mod 4} = (A^0_i)_2 = 2
\]

For the second element, \(R^1_i\), the set \(A^1_i\) is \(\{0, 1, 3\}\), as the element \(R^0_i = 2\) was removed from it. Therefore the second element of \(R_i\) becomes:

\[
A^1_i = \{0, 1, 3\} \quad R^1_i = (A^1_i)_{2 \mod 3} = (A^1_i)_2 = 3
\]

\(^4\) This isn’t always the case. Previous variations of the GAuGE system [9] have used a different mapping process at this stage.
The third element is calculated in the same fashion:

\[ A^2_i = \{0, 1\} \quad R^2_i = (A^2_i)_{4 \mod 2} = (A^2_i)_0 = 0 \]

And finally the fourth element:

\[ A^3_i = \{1\} \quad R^3_i = (A^3_i)_{5 \mod 1} = (A^3_i)_0 = 1 \]

So the string of positions becomes:

\[ R_i = (2, 3, 0, 1) \]

The string of values \( \hat{R}_i \) can then be calculated using Eq. (3):

\[
\begin{align*}
\hat{R}^0_i &= \hat{X}^0_i \mod 2 = 1 \mod 2 = 1 \\
\hat{R}^1_i &= \hat{X}^1_i \mod 2 = 2 \mod 2 = 0 \\
\hat{R}^2_i &= \hat{X}^2_i \mod 2 = 3 \mod 2 = 1 \\
\hat{R}^3_i &= \hat{X}^3_i \mod 2 = 1 \mod 2 = 1 
\end{align*}
\]

Finally, by using Eq. (4), the phenotype string \( P_i \) can be constructed in the following manner:

\[
\begin{align*}
P^2_i &= 1 \\
P^3_i &= 0 \\
P^0_i &= 1 \\
P^1_i &= 1 
\end{align*}
\]

So the phenotype string \( P_i \), ready for evaluation, is:

\[ P_i = 1110 \]

### 4 Functional Dependency and Degenerate Code

In this section, the functional dependency occurring within the elements of the genotype string is analysed, along with the effect of degenerate code on that dependency. By functional dependency, the manner in which the function of a gene is (possibly) affected by the function of previous genes is meant. As shown in Eq. 2, each position specification derived from the genotype string depends on previous position specifications, and on its location within the genotype string.

The degenerate code feature refers to the many-to-one mapping process from genotype to phenotype. As seen in Section 3.1, the use of the mod function maps the value specified by \( X^j_i \) onto the set of available positions left on the phenotype string.

In order to characterise the dependency of each \( R^j_i \) on the elements \( R^0_i \ldots R^{j-1}_i \) and on \( j \) itself, the mean value of each \( R^j \) is computed. If this dependency did not exist, then this mean value would be constant in \( j \). Section 4.1 shows that this mean value depends on \( j \), and section 4.2 shows the effect of the introduction of degeneracy on this mean value.
Problem length ($l$): 128
Population size ($N$): 100000
Position field size ($pfs$): 7 bits
Value field size ($vfs$): 1 bit

Table 1. Experimental setup. 7 bits were used for the position field, as this is the minimum number of bits required to encode 128 positions

4.1 Computing the mean of $R^j$

For every $j \in \{0, \ldots, \ell - 1\}$, the mean (or expectation) of $R^j$ is denoted $E(R^j)$. This expectation can be computed exactly in $\ell!$ operations (see Remark 1). It is thus crucial to use another method to compute the expectation $E(R^j)$ of all $R^j_i$. The method chosen here is the numerical simulation of the expectation with the Monte Carlo method [7], whose main interest is to give a confidence interval for the approximation of the mean.

The principle of the Monte Carlo simulation is as follows: in order to calculate the average position $E(R^j)$ of all $R^j_i$, a population of $N$ individuals (with $N$ large, typically equal to 10000) is generated. The $N$ individuals are denoted $(R^j_i)_{1 \leq i \leq N}$. Each mean of $R^j$, $E(R^j)$ is approximated by

$$
\frac{1}{N} \sum_{i=1}^{N} R^j_i.
$$

(5)

A consequence of the Central Limit Theorem is that there is a confidence interval for this approximation. More precisely, for the 99% interval confidence, there is a probability equal to 0.99 that the true values of the average position $E(R^j)$ are within the interval

$$
\left[ \frac{1}{N} \sum_{i=1}^{N} R^j_i - \alpha_i(N), \frac{1}{N} \sum_{i=1}^{N} R^j_i + \alpha_i(N) \right]
$$

where $\alpha_i(N)$ is equal to

$$
2.58 \sqrt{\frac{1}{N} \sum_{i=1}^{N} (R^j_i)^2 - \left( \frac{1}{N} \sum_{i=1}^{N} R^j_i \right)^2}.
$$

(6)

The values used in the simulations are given in Table 4.1. Figure 1 shows a plot of those averages, along with the 99% interval.

Experimental results The graph illustrates the functional dependency across the elements of all the strings of positions $R$, from the left to the right side. As each number specified by $X^j_i$ is modded by smaller values (128, 127, ...), the index it specifies in the set $A^j_i$ is wrapped around to the beginning of that set.
Fig. 1. Average position specification per element of $X_i$, for a genotype generating a phenotype of length 128. The $x$-axis shows each position (index $j$) of the string $X_i$, and the $y$-axis the average position it specifies, $E(R_j^i)$ (results averaged over 100000 randomly created individuals)

(due to the use of the mod operator), which means that smaller values will be specified on average. However, as the number to mod $X_j^i$ by reaches the value 64, the distribution of values specified is again uniform, which explains the return of the average position specified to a more average value again (close to 63.5).

This balancing phenomenon continues on, creating a *ripple effect* [12] across the positions specified in each $R_i$. The slight upward slope in the positions specified is explained by the fact that between the points where the positions return to close to average values (0, 64, 85, 96, 102, 106, …), the positions specified are smaller than the average.

### 4.2 Introducing degeneracy

In this section, the effect of the introduction of degeneracy at the genotype level is investigated. Degeneracy in the position specifications is introduced by using bigger values for $pfs$; indeed, this introduces a redundant mapping, as different values for $X_j^i$ will map to the same $R_j^i$ value (through the use of the $(\mod)$ function). This will have the effect of balancing the position specifications across

---

5 This sequence of peaks represents the points where the mod operator wraps around (e.g., 128 mod 64 = 0, and 128 mod 43 = 42, but 128 (mod 4)2 = 2 (128 − 43 = 85)
each $R_i$ string, as each element of that string will be averaged across a larger set of values, thus flattening the ripples observed.

A theoretical proof of the fact that introduction of degeneracy flattens the ripples, and that, consequently, the position each $R_i^j$ specifies becomes less dependent from $j$, can be given. Specifically we shall prove in Appendix A the following Theorem

**Theorem 1.** Let $i$ be an integer in $\{0, ..., N-1\}$ and let $R_i = (R_i^0, ..., R_i^{\ell-1})$ defined in Section 3 by Eq. (2), then for all $j \in \{0, ..., \ell-1\}$

$$E(R_i^j) = \frac{\ell - 1}{2} + \mathcal{O}(\ell^3 2^{-pfs})$$

(7)

where the notation $\mathcal{O}(\ell^3 2^{-pfs})$ means that $\frac{\mathcal{O}(\ell^3 2^{-pfs})}{2^{2-pfs}}$ is bounded when $pfs \to \infty$ by a constant independent of $\ell$. A consequence of Eq. (7) is thus

$$E(R_i^j) \xrightarrow{pfs \to \infty} \frac{\ell - 1}{2}$$

for all $j$

meaning that the expectation $E(R_i^j)$ becomes constant and thus independent of the index $j$ when $pfs$ is high. Moreover $\mathcal{O}(\ell^3 2^{-pfs})$ gives a quantitative idea of the value of $pfs$ in order to consider that $E(R_i^j)$ can be approximated by $\frac{\ell - 1}{2}$; this is precisely when $\frac{2^{pfs}}{\ell} \gg 1$.

To illustrate these findings, a similar experimental setup to that used in Section 4.1 is used; in these experiments, the value of $pfs$ varied from 7 bits to 14 bits. Figure 2 plots the results of the simulation.

**Experimental results** The introduction of degeneracy has an interesting effect. By using a larger number of bits for $pfs$, the range of numbers each $X_i^j$ can specify is increased (e.g., using 14 bits, the range will be 0-16383, rather than 0-127 with 7 bits per gene); when modded by the number of available positions (128, 127, ...), this reduces significantly the differences between the position specified by each field, thus effectively flattening the ripples. Figure 2 illustrates this effect: by gradually increasing the value of $pfs$, the differences in the average of each $R_i^j$ are reduced, as is the slight increasing average position specified toward the end of each $R_i$ string.

5 Conclusions and future work

This paper has presented a detailed investigation of the mapping process employed in the GAuGE system. A formal definition of that process has been given, and is used to characterise the functional dependency occurring between the position specifications of the genotype string, and the effect that the introduction of degeneracy has on that dependency. The theorem presented, backed up by experimental evidence, shows that the introduction of degeneracy reduces the functional dependency between the position specifications and their placement...
Detailed analysis of the GAuGE system 480009

Average position specifications - Length 128 - 100000 tests

Fig. 2. Effect of degeneracy on the average position specification per element of $X_i$. The $x$-axis shows each position ($j$) of the string $X_i$, the $y$-axis (depth) shows the $pfs$ value chosen, and the $z$-axis (vertical) shows the average position specified, $E(R^j_i)$ (results averaged over 100000 randomly created individuals)

on the real specifications string, and illustrates some of the implications of using degenerate code at the genotype level.

The models presented can be of use when analysing alternative mapping processes, not only for the GAuGE system, but also for other systems where a detailed analysis of functional dependency should be required.

As can be expected, when the evolutionary process starts, the distribution of position specifications across the genotype is no longer uniform. Future work will therefore analyse the implications that the results reported have during the evolution process.

A Appendix: Proof of Theorem 1

Before giving the proof of Theorem 1, an alternative view of the mapping described in Section 3.1 is proposed. The mapping is seen as a permutation of the set of values $\{R^0_i, \ldots, R^{\ell-1}_i\}$. Indeed, the string of positions $R_i = (R^0_i, \ldots, R^{\ell-1}_i)$ where for all $j \in \{0, \ldots, \ell - 1\}$, $R^j_i \in \{0, \ldots, \ell - 1\}$ is nothing but a classical manner to write a permutation of $\ell$ elements, i.e. a bijective function of a set of $\ell$ elements into itself. Formally the permutation $\rho : \{0, \ldots, \ell - 1\} \rightarrow \{0, \ldots, \ell - 1\}$ induced by the mapping is defined as follows:
\[ \rho(j) = R_i^j \text{ for } 0 \leq j \leq \ell - 1 \]

And the mapping consists then in permuting the elements \( \{ R_i^0, \ldots, R_i^{\ell-1} \} \) into \( \{ R_i^{\rho(0)}, \ldots, R_i^{\rho(\ell-1)} \} \). This alternative view is used in the proof of the following Theorem:

**Theorem 1.** Let \( i \) be an integer in \( \{0, \ldots, N - 1\} \) and let \( R_i = (R_i^0, \ldots, R_i^{\ell-1}) \) defined in Section 3 by Eq. (2), then for all \( j \in \{0, \ldots, \ell - 1\} \)

\[
E(R_i^j) = \frac{\ell - 1}{2} + O(\ell^3 2^{-pf_s})
\]

**Proof.** Considering each \( R_i^j \) defined by Eq. 2 as a random variable, one sees that the distributions of \( R_i^j \) are the same for any \( i \). Thus, for sake of simplicity, the subscript \( i \) is omitted in this proof. For consistency with the formalisation of the mapping in terms of permutation, where \( R = (R_0^0, \ldots, R_{\ell-1}) \) is identified with a permutation, for any given permutation \( \rho \in S_\ell \) (\( S_\ell \) being the permutation group on \( \{0, \ldots, \ell - 1\} \)) the notation \( R_\rho \) denotes in the sequel the event:

\[ R_\rho = \rho(j) \text{ for all } j \]

From the construction of \( R_\rho \) one sees that, for an acceptable value \( a \) (among the \( \ell - j \) possible values) for \( R_\rho \), one has

\[
P(R_\rho = a | R_\rho^{-1}, \ldots, R_\rho^0) = 2^{-pf_s} \left( \left\lfloor \frac{2pf_s}{\ell - j} \right\rfloor + \mathbb{1}_{a < 2pf_s \pmod{\ell - j}} \right)
\]

where \( \left\lfloor \frac{2pf_s}{\ell - j} \right\rfloor \) denotes the integer part of \( \frac{2pf_s}{\ell - j} \) and \( \mathbb{1}_{a < 2pf_s \pmod{\ell - j}} = 1 \) if \( a < 2pf_s \pmod{\ell - j} \) and 0 otherwise. Simplifying Eq. (8), one has,

\[
P(R_\rho = a | R_\rho^{-1}, \ldots, R_\rho^0) = \frac{1}{\ell - j} + r_{j, pf_s}(a)
\]

with \( |r_{j, pf_s}(a)| \leq 2^{-pf_s} \). Therefore, for a given permutation \( \rho \), \( P(R = \rho) \) is the product of the above probabilities and a straightforward computation leads to

\[
P(R = \rho) = \frac{1}{\ell!} \left( 1 + O(\ell^2 2^{-pf_s}) \right).
\]

where the notation \( O(\ell^2 2^{-pf_s}) \) means that \( \frac{O(\ell^2 2^{-pf_s})}{\ell^2 2^{-pf_s}} \) is bounded when \( pf_s \to \infty \) by a constant independent of \( \ell \). Now the expectation of the random variable \( R_\rho \) is

\[
E(R_\rho) = \sum_{0 \leq a < \ell} a \sum_{\rho \in S_\ell} P(R = \rho \& \rho(j) = a)
\]

(9)
but the number of permutations $\rho$ such that $\rho(j)$ is fixed and is equal to $(\ell - 1)!$, hence

$$E(R^j) = (\ell - 1)! \frac{1}{\ell!} (1 + O(\ell^2 2^{-pfs}))(0 + 1 + \cdots + (\ell - 1)).$$

Finally

$$E(R^j) = (\ell - 1)! (1 + O(\ell^2 2^{-pfs}))$$

or

$$E(R^j) = \frac{\ell - 1}{2} + O(\ell^3 2^{-pfs}).$$

\[ \square \]

Remark 1. Complexity for the computation of $E(R^j)$ As one can see on Eq. (9), the complexity to compute $E(R^j)$ is $\ell!$.

Acknowledgments

The authors would like to thank all anonymous reviewers, whose suggestions led to several improvements of this work; in particular, the reviewer whose clever interpretation of the mapping in term of permutations led to a more precise estimation in the theorem given and to simplifications in the proof. We would also like to thank the feedback provided by Zbigniew Skolicki on the explanation of the ripple effect.

References