Grammar-based Evolution of Polyominoes

Jessica Mégane[1 [0000−0001−6697−5423]], Eric Medvet[2 [0000−0001−5652−2113]], Nuno Lourenço[1 [0000−0002−2154−0642]], and Penousal Machado[1 [0000−0002−6308−6484]]

[1] University of Coimbra, CISUC/LASI - Centre for Informatics and Systems of the University of Coimbra, Department of Informatics Engineering
[2] Department of Engineering and Architecture, University of Trieste, Italy
{jessicac,naml,machado}@dei.uc.pt, emedvet@units.it

Abstract. Languages that describe two-dimensional (2-D) structures have emerged as powerful tools in various fields, encompassing pattern recognition and image processing, as well as modeling physical and chemical phenomena. One kind of two-dimensional structures is given by labeled polyominoes, i.e., geometric shapes composed of connected unit squares represented in a 2-D grid. In this paper, we present (a) a novel approach, based on grammars, for describing sets of labeled polyominoes that meet some predefined requirements and (b) an algorithm to develop labeled polyominoes using the grammar. We show that the two components can be used for solving optimization problems in the space of labeled polyominoes, similarly to what happens for strings in grammatical evolution (and its later variants). We characterize our algorithm for developing polyominoes in terms of representation-related metrics (namely, validity, redundancy, and locality), also by comparing different representations. We experimentally validate our proposal using a simple evolutionary algorithm on a few case studies where the goal is to obtain a target polyomino: we show that it is possible to enforce hard constraints in the search space of polyominoes, using a grammar, while performing the evolutionary search.

Keywords: polyomino, grammar, representation, 2-D patterns

1 Introduction

Two-dimensional (2-D) languages have emerged as powerful tools in various fields, initially motivated by problems in pattern recognition and image processing [4, 6, 14]. These languages generate 2-D objects and can be designed to generate either a simple rectangle or a more intricate shape such as a decagon. One shape that has garnered significant interest due to its unique properties [1, 5, 13] and wide-ranging applicability [8, 15, 21, 25, 26, 28, 29, 34] is the polyomino.

Polyominoes [7] are geometric shapes composed of connected unit squares, forming a finite set of cells within a 2-D grid. These shapes are also commonly referred to as lattice animals in the physical [8] and chemical [26] fields, where

they have been popularly used to model branched polymers, molecules and percolation processes, providing valuable insights into the behavior of these complex systems [3, 37]. The field of combinatorial optimization and mathematics has extensively explored polyominoes, due to their rich mathematical properties [1, 5, 13]. Similarly, the theoretical formal language domain has been inspired by their practical usages.

In these fields, it is often crucial to find one or more polyominoes that maximize specific objectives while satisfying predefined structural requirements. Two powerful mechanisms for describing and generating 2-D shapes are grammars [15, 25, 27, 29, 38].

Grammars define a language with a set of rules that can impose certain constrains. These grammars can be designed either as one-dimensional encodings of two-dimensional structures [38] or be two-dimensional structured representations [27, 29]. To the best of our knowledge, there exists only a single approach that utilizes grammars to define polyominoes constraints and then attempts to optimize them [38], namely, for finding an assembly of identical polyominoes. While this represents the only usage of a grammar to define polyominoes in the literature, numerous examples exist where grammars are employed to generate 2-D pictures [12, 14, 18, 35], finding practical applications in popular tasks such as mathematical formula recognition [15, 25, 29].

Polyominoes can be further enhanced by assigning labels to individual cells, providing additional information for each cell within the structure. In this paper, we present a novel approach for generating labeled polyominoes that meet some structural requirements defined in a formal way: for this purpose, we (a) define the concept of polyomino context-free grammars (PoCFGs) as an extension of context-free grammars (CFGs) and (b) propose a development algorithm that can be used for generating a polyomino adhering to a PoCFGs. Our proposed algorithm constructs these polyomino structures with precise control of the shape of the polyomino and labeling of its cells. When used inside an evolutionary algorithm (EA), it allows solving optimization problems over the space of labeled polyominoes adhering to a given PoCFG. This process only requires the provision of a grammar and a fitness function to the selected EA; notably, it does not require users to provide variation operators that guarantee that varied polyominoes will still adhere to the PoCFG—i.e., operators with the closure property. This fact greatly increases the applicability of our approach, lowering the barrier to polyominoes optimization, much like grammatical evolution (GE) [31] did with regular languages.

Since our algorithm is greatly agnostic with respect to the genotypic representation employed by the EA, we compare different representations in terms of representation-related metrics (validity, redundancy, and locality). Moreover, to showcase the effectiveness of our approach, we evolve some polyominoes in a few case studies where the goal is to evolve a polyomino adhering to a grammar that is as much as possible similar to a pre-defined target polyomino. We show experimentally that evolutionary optimization does work, giving polyominoes that

are more and more similar to the target one while all adhering to the provided grammar, i.e., meeting the user-defined constraints.

2 Our proposal: describing and evolving polyominoes

2.1 Labeled polyomino

A *polyomino* is a 2-D geometric figure formed by one or more squares (or *cells*) joined together along their sides. A *labeled polyomino* defined over an alphabet A is a polyomino in which each cell is associated with exactly one symbol (or *label*) $a \in A$. For brevity, from now on we will write simply polyomino for referring to labeled polyominoes. We denote by $\mathcal{P}_A$ the set of all the polyominoes defined over A.

By assigning a coordinate $(x_0, y_0) \in \mathbb{Z}^2$ to one of the cells of a polyomino p, we denote by $p_{x,y} \in A$ the label of the cell of p displaced by $x - x_0$ cells along the x-axis and $y - y_0$ cells along the y-axis with respect to the cell at (x_0, y_0). We write $p_{x,y} = \varnothing$ if there is no cell at (x, y) in p.

A *referenced polyomino* is a polyomino in which one cell is identified as *reference cell*. By convention, we assume that the reference cell is assigned to the coordinate $(0, 0)$.

2.2 Polyomino context-free grammar (PoCFG)

A polyomino context-free grammar (PoCFG) $\mathcal{G}$ is a tuple $\mathcal{G} = (N, T, n_1, \mathcal{R})$ where N is a finite set of *non-terminal* symbols, T is a finite set of *terminal* symbols, with $N \cap T = \emptyset$, $n_1 \in N$ is the starting symbol (or *axiom*), and $\mathcal{R}$ is a finite set of *production rules*. A production rule is a pair composed of a non-terminal symbol (the left-hand-side of the rule) and a referenced polyomino defined over the alphabet $N \cup T$ (the right-hand-side).

Similarly to the case of CFGs for strings, we represent a PoCFG with a compact notation which resembles the Backus-Naur form (BNF). In BNF rules are grouped together by their non-terminal symbol and the first rule is the one for the starting symbol n_1. Figure 2 shows an example of five PoCFGs in BNF (the ones used in our experiments).

As for the case of CFGs for strings, a PoCFG $\mathcal{G} = (N, T, n_1, \mathcal{R})$ is a compact way for defining a (possibly infinite) set of polyominoes defined over T; we denote by $\mathcal{P}_\mathcal{G} \subseteq \mathcal{P}_T$ the set of polyominoes defined by $\mathcal{G}$. In the next section, we describe a constructive process that allows to obtain one $p \in \mathcal{P}_\mathcal{G}$. Note that the problem of deciding whether a given polyomino p belongs to $\mathcal{P}_\mathcal{G}$ is beyond the scope of this paper—for CFGs that meet some requirements, this problem is solvable for the case of strings [32].

2.3 PoCFG-based development algorithm

We propose a *development algorithm* for obtaining a polyomino $p \in \mathcal{P}_\mathcal{G}$ for a PoCFG $\mathcal{G}$. We call it development algorithm because it iteratively modifies a

polyomino by either adding new cells or modifying existing ones according to the production rules of $\mathcal{G}$ and starting from a single cell polyomino given by the axiom—from this point of view, it resembles a developmental process.

Design principles. Since the eventual usage of this algorithm is within the process of the evolutionary optimization over $\mathcal{P}_\mathcal{G}$, we designed it to receive as input a source of information that is used for choosing which production rules to apply. Consistently with the intended usage, we call this input the *genotype* g. The development algorithm hence maps a genotype g to a polyomino p.

We designed our algorithm to be largely agnostic to the kind of genotype being fed as input, i.e., to $G \ni g$. We achieved this goal by making the algorithm modular, i.e., by decoupling the part where a suitable production rule is chosen using g from the rest of the algorithm, namely from the part in which the chosen rule is used and the part in which suitable rules are identified. Indeed, in Section 3.2 we experimentally compare realizations of the development algorithm where the genotype is a bit- or an integer-string or a more complex data structure.

Working. Algorithm 1 presents our development algorithm in the form of pseudocode. It takes as input the genotype $g \in G$ and the PoCFG $\mathcal{G}$ and, as parameters, a sorting criterion c and an overwriting flag o; it returns either a polyomino $p \in \mathcal{P}_\mathcal{G}$ or $\varnothing$ if it is not possible to develop a polyomino with the given inputs and parameters.

The algorithm works as follows. First, it sets (line 2) p to the one-cell polyomino with the only cell being labeled with the axiom n_1. Then, it iteratively modifies p according to these steps: (i) it finds (line 5) all the cells in p that are labeled with a non-terminal symbols in N, i.e., those which can be replaced according to a rule in $\mathcal{R}$; (ii) it chooses (line 9) one cell $(x^\star, y^\star)$ to be the target of the replacement using the sorting criterion c; (iii) it chooses (line 11), based on the genotype g and the state s (initialized to $\varnothing$, see Section 2.3), one rule to apply among the ones suitable for the cell at $(x^\star, y^\star)$; (iv) finally, if possible, it performs (line 18) the replacement in p according to the chosen rule. The last step, i.e., the actual modification of p, consists in "putting" the referenced polyomino p' "over" p with the reference cell of p' placed at $(x^\star, y^\star)$ in p. The iterations stop if (a) no more cells labeled with a non-terminal symbol are present in the polyomino or (b) some of the steps cannot be performed (see below).

Step (ii) above (FIRST() in Algorithm 1) corresponds in selecting one non-terminal cell to be the target of the replacement—in general, there can be more than one non-terminal cell in the polyomino being developed at some point of the mapping. The way this choice is made is important because it can impact on whether other steps fail, due to production rules not being applicable (see below for those conditions). We cast the problem of choosing one cell as a sorting problem and we explore three sorting criteria, according to which we select (a) (*Position* criterion) the non-terminal cell with the lowest y-coordinate in p

Algorithm 1 Algorithm to generate a polyomino $p \in \mathcal{P}_\mathcal{G} \cup \{\varnothing\}$ from a genotype $g \in G$ using a PoCFG $\mathcal{G}$, a sorting criterion c, and an overwriting flag o.

```
1: function DEVELOP(g, G; c, o)
2:     p ← SINGLE(STARTINGSYMBOL(G))                    ▷ init with starting symbol
3:     s ← ∅
4:     while true do
5:         {(x_i, y_i)}_i ← NONTERMINALCELLS(p)
6:         if |{(x_i, y_i)}_i| = 0 then                  ▷ no non terminal cells
7:             break
8:         end if
9:         (x*, y*) ← FIRST(R; c)                        ▷ find cell to be replaced
10:        R_n ← OPTIONSFOR(p_{x*,y*}, G)  ▷ find replacing ref. pol. for p_{x*,y*} ∈ N in G
11:        (p', s) ← CHOOSE(R_n, g, s)     ▷ choose one replacing referenced polyomino
12:        if p' = ∅ then                                ▷ no chosen replacing polyomino
13:            return ∅
14:        end if
15:        if ¬o ∧ ¬FITS(p', p, x*, y*) then     ▷ cannot replace p with p' at x*, y*
16:            return ∅
17:        end if
18:        p ←REPLACE(p, p', x*, y*)
19:    end while
20:    return p
21: end function
```

and, in case of, tie, the one with the lowest x-coordinate; or (b) (*Recency* criterion) the one which has been inserted in p most recently (i.e., at the most recent iteration of the algorithm) and, in case of tie, the one selected with the Position criterion; or (c) (free *Sides* criterion) the one which has most free sides, i.e., sides on which there are no other cells, and, in case of tie, the one selected with the Position criterion. In Algorithm 1, the parameter c represents the sorting criterion determining the behavior of FIRST(). In Section 3.1 we compare experimentally the variants of the algorithm obtained with different criteria.

Step (iii) is the one where a production rule among the suitable ones for the target cell is chosen based on the genotype, through the CHOOSE() function. We describe three alternatives for this function in Section 2.3.

No-mapping conditions. There are two conditions according to which the development algorithm returns $\varnothing$, i.e., fails in mapping a genotype g to a polyomino $p \in \mathcal{P}_\mathcal{G}$. First, if it is not possible to choose a rule for a given replaceable cell, given a genotype g, i.e., if CHOOSE$(\mathcal{R}_r, g)$ returns $\varnothing$: this condition usually (see next section) corresponds to the case where g has been completely consumed. We introduced this possibility as a mechanism for avoiding endless execution of the iterative part of the development algorithm—similar mechanisms indeed exist in most of the variants of GE, such as structured grammatical evolution (SGE) [16] and weighted hierarchical grammatical evolution (WHGE) [2].

Second, if the chosen cell at $(x^\star, y^\star)$ of p is not replaceable using the referenced polyomino p' constituting the right-hand-side of the chosen rule, then the mapping fails. A cell $(x^\star, y^\star)$ of p is not replaceable with p' if some cells of p close to the cell are not empty and should be replaced by a corresponding cell in p'. We remark that this condition is peculiar of the case of polyominoes and does not have a counterpart in plain strings, differently from the previous one. In facts, while in the case of strings one can replace a single symbol with a sequence of symbols simply by "enlarging" the gap between the leading and trailing (with respect to the symbol being replaced) substrings, this accommodation is not possible in 2-D. In Algorithm 1, this condition is verified by FITS().

In this work, we also explore a variant of the development algorithm in which a rule is always applicable, that is, a referenced polyomino p' can always be placed at a given $(x^\star, y^\star)$, regardless of the fact that close cells are empty. In other words, in this variant we allow for *overwriting* while applying production rules—in Algorithm 1, the Boolean parameter o represents an overwriting flag.

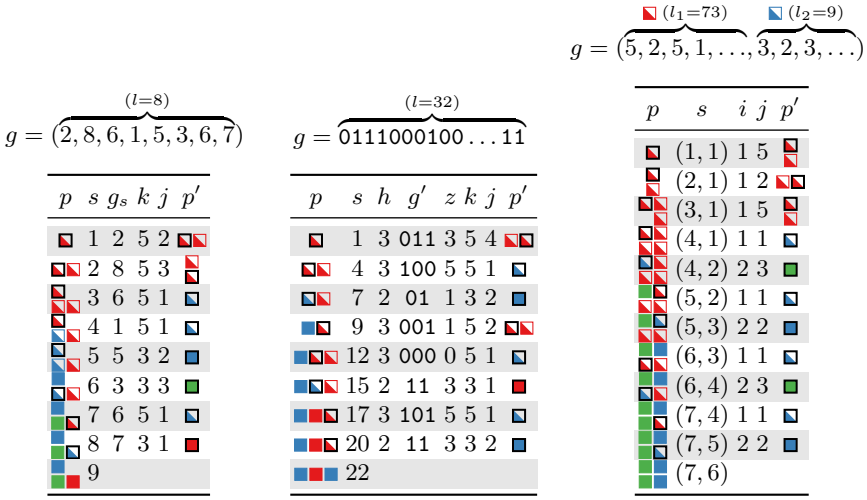
Different representations. We designed our development algorithm to accommodate different *representations*, i.e., different domains G for the genotype $g \in G$. Our rationale is twofold. First, we wanted to show that the algorithms is general, thanks to the decoupling of the choice of the production rule and the rest of the process. Second, we wanted to build on previous research and practice on the similar case of grammar-guided genetic programming (G3P), where different kinds of genotype (and different ways of using it) have been proposed to improve the general effectiveness of the evolutionary search, e.g., bit-strings in the early GE and WHGE, structured strings of integers in SGE.

We considered four representations; thanks to the modular nature of the algorithm, they correspond to four implementations of the CHOOSE() function of Algorithm 1. In all cases, we assume that the procedure for choosing a production rule given the genotype is stateful, that is, that subsequent invocations with the same g may give different outputs. We formalize this assumption by including a state s as argument for CHOOSE() and by making the function return a new state s, along with the chosen reference polyomino p'. We remark that the state is initialized to an empty state $\varnothing$ at every new execution of the development algorithm. The domain of the state depends on the representation.

Figure 1 shows an example of the execution of the development algorithm with three of the four representations described in detail below.

String of integers. In this representation, a genotype g is an l-long string of integers, i.e., $G = \{1, \ldots, b\}^l \subseteq \mathbb{N}^l$, and the state s is a integer, i.e., $S = \{1, \ldots, l\} \in \mathbb{N}$, used as a counter.

Given a genotype $g = (g_1, \ldots, g_l)$, the production rules $\mathcal{R}_n = (r_1, \ldots, r_k)$ for the non-terminal n (where each r_j is a pair (n, p_j), with p_j a referenced polyomino defined over $N \cup T$), and the state s, the function CHOOSE() for this representation works as follows. Concerning the state, if s is $\varnothing$, s becomes 1, otherwise s becomes $s + 1$. Concerning the output reference polyomino p', if $s > l$, $p' = \varnothing$, otherwise $p' = p_j$, with $j = ((g_s - 1) \bmod k) + 1$.



p	s	g_s	k	j	p'
	1	2	5	2	
	2	8	5	3	
	3	6	5	1	
	4	1	5	1	
	5	5	3	2	
	6	3	3	3	
	7	6	5	1	
	8	7	3	1	
	9				

(a) With ints(8, 8)

p	s	h	g'	z	k	j	p'
	1	3	011	3	5	4	
	4	3	100	5	5	1	
	7	2	01	1	3	2	
	9	3	001	1	5	2	
	12	3	000	0	5	1	
	15	2	11	3	3	1	
	17	3	101	5	5	1	
	20	2	11	3	3	2	
	22						

(b) With bits(32)

p	s	i	j	p'
	(1, 1)	1	5	
	(2, 1)	1	2	
	(3, 1)	1	5	
	(4, 1)	1	1	
	(4, 2)	2	3	
	(5, 2)	1	1	
	(5, 3)	2	2	
	(6, 3)	1	1	
	(6, 4)	2	3	
	(7, 4)	1	1	
	(7, 5)	2	2	
	(7, 6)			

(c) With structured(82, 2)

Fig. 1: Example of the development of a polyomino with the grammar of Figure 2b and three representations (one table for each representation). Each row in the table represents one iteration of the algorithm (with the Position sorting criterion and no overwriting). The thick black border denotes in p the cell that is being replaced and in p' the reference cell.

Intuitively, here CHOOSE() consumes the genotype one integer at once and chooses the rule using the mod rule, as in the original GE.

We denote this representation with ints(l, b), b being the maximum value each genotype element may assume and l being the genotype length. That is, l and b are parameters for this parametric representation.

String of bits. In this representation, $G = \{0, 1\}^l$ and $S = \{1, \ldots, l\} \in \mathbb{N}$.

Given $g = (g_1, \ldots, g_l)$, $\mathcal{R}_n$, and s, CHOOSE() works as follows. Concerning the state, if s is $\varnothing$, s becomes 1, otherwise s becomes $s + h$, with $h = \lceil \log_2 |\mathcal{R}_n| \rceil$. Concerning p', if $s + h > l$, $p' = \varnothing$, otherwise it (i) takes the h bits $g' = (g_s, g_{s+1}, \ldots, g_{s+h})$ in g, (ii) converts them to an integer $z \in \{1, \ldots, 2^h\}$, then (iii) returns $p' = p_j$, with $j = (z \bmod k) + 1$.

Intuitively, here CHOOSE() consumes the genotype h bits at once, with h the smallest possible to accommodate $\mathcal{R}_n$ (possibly not-consuming bits if $|\mathcal{R}_n| = 1$), and chooses the rule using the mod rule on the bit-to-integer conversion of the consumed h bits.

We denote this representation with bits(l).

String of reals. In this representation, $G = \mathbb{R}^l$ and $S = \{1, \ldots, l\} \in \mathbb{N}$.

Given $g = (g_1, \ldots, g_l)$, $\mathcal{R}_r$, and s, CHOOSE() works as follows. Concerning the state, it works as ints(n, l). Concerning p', if $s > l$, $p' = \varnothing$, otherwise it

(i) clamps down g_s in $[0, 1]$, as $h = \min(1, \max(0, g_s))$, then (ii) returns $p' = p_j$, with $j = \max(1, \lceil k g_s \rceil)$.

Intuitively, here CHOOSE() consumes the genotype one real value at once and chooses the rule based on the value clamped in $[0, 1]$ and mapped to $\{1, \ldots, |\mathcal{R}_n\}$.

We denote this representation with reals(l).

Structured string of integers. In this representation, a genotype is a set of strings of integers, one string for each non-terminal in the grammar, and the state is a set of counters, one for each non-terminal. Let $N = \{n_1, \ldots, n_m\}$ be the set of non-terminals and let $\mathcal{R}_{n_j} \subseteq \mathcal{R}$ be the set of production rules for the non-terminal n_j. Formally, $G = \{1, \ldots, |\mathcal{R}_{n_1}|\}^{l_1} \times \cdots \times \{1, \ldots, |\mathcal{R}_{n_m}|\}^{l_m}$ and $S = \{1, \ldots, l_1\} \times \cdots \times \{1, \ldots, l_m\}$, with $\sum_{j=1}^{j=m} l_j = l$.

We determine the number l_j of genotype elements for the j-th non-terminal of the grammar, based on the overall genotype length l, as follows. (i) We start from a bag $\mathcal{N} = \{n_1\}$ of non-terminal symbols containing the axiom only. (ii) We repeat for n_{rec} times this procedure: for each element n of $\mathcal{N}$, we consider all the rules $\mathcal{R}_n$ for n, we take all the referenced polyominoes appearing on the right-hand-side, and we add to $\mathcal{N}$ all the non-terminal symbols appearing in them (possibly multiple times). (iii) Finally, based on the content of $\mathcal{N}$, we reserve to each non-terminal n a proportion of genotype elements based on the amount of n items in $\mathcal{N}$, namely, for n_j we set $l_j = \left\lfloor l \frac{|\{n \in \mathcal{N} : n = n_j\}|}{|\mathcal{N}|} \right\rfloor$ (reasonably adjusted to have $\sum_{j=1}^{j=m} l_j = l$). The rationale of this procedure is to have a number of genotype elements suitable for performing "enough" productions with each given non-terminal, while still constraining the genotype to be l-long. The parameter n_{rec} determines how much non-terminals that are, intuitively, more recursive in the grammar take more space in the genotype.

Given $g = (g_{1,1}, \ldots, g_{1,l_1}, \ldots, g_{m,1}, \ldots, g_{m,l_m})$, the rules $\mathcal{R}_{n_i}$ for a non-terminal n_i, and $s = (s_1, \ldots, s_m)$ (or $s = \varnothing$), CHOOSE() for this representation works as follows. Concerning the state, if $s = \varnothing$, then s becomes $(1, \ldots, 1)$ (i.e., a m-long vector of ones); otherwise, if $s = (s_1, \ldots, s_m)$, then $s_i = s_i + 1$. Concerning p', if $s_i > l_i$, $p' = \varnothing$, otherwise $p' = p_j$, with $j = g_{i,s_i}$.

Intuitively, here CHOOSE() consumes the genotype one integer value at once in the portion of the genotype corresponding to the non-terminal being replaced and chooses the rule based on the "current" genotype element. Note that the mod rule here is not needed, since the domain of each genotype part exactly matches the number of rules for the corresponding non-terminal symbol.

We denote this representation with structured(l, n_{rec}).

2.4 Evolution of polyominoes

Having defined a way to map a genotype $g \in G$ to a polyomino $p \in \mathcal{P}_\mathcal{G}$ for a grammar $\mathcal{G}$, we can solve problems of optimization over $\mathcal{P}_\mathcal{G}$ using evolutionary computation (EC), that is, with an EA. Since we defined mapping variants for different kinds of genotype, we might use any EA that is suitable for the corresponding G, e.g., evolutionary strategy (ES) [9], or maybe the more recent

OpenAI-ES [33], for the reals(l) representation and a genetic algorithm (GA) for bits(l), possibly including a linkage-exploitation mechanism [36] as done in [20]. However, for simplicity, in this work we experiment with a single EA, described below, and we leave the investigation of other EAs as future work.

Given a fitness function $f : \mathcal{P}_\mathcal{G} \to \mathbb{R}$ (we assume to tackle minimization problems, without loss of generality), we evolve polyominoes with a simple EA with two variation operators (mutation and crossover, each being representation-specific), tournament selection for parents selection, and overlapping between parents and offspring. In detail, we first initialize (with a representation-specific procedure) a population P of n_{pop} individuals. Then, we repeat n_{gen} times the following steps: (i) we generate $r_{\text{x-over}} n_{\text{pop}}$ new individuals with crossover, i.e., each one by selecting two parents from P with tournament selection (with n_{tour} size) and applying them a crossover operator; (ii) we generate $(1 - r_{\text{x-over}}) n_{\text{pop}}$ new individuals with mutation, selecting the parent with tournament selection; (iii) we merge all newly generated individuals to the parents, hence obtaining a population P with $2n_{\text{pop}}$ individuals; (iv) we trim P with truncation selection, retaining the best n_{pop} individuals according to the fitness function f. At the end, we return the individual, i.e., the polyomino, with the lowest fitness.

Concerning the representation-specific initialization procedure, we simply generate each genotype g by sampling each one of its element in the proper domain with uniform probability, i.e., in $\{1, \dots, b\}$ for ints(l, b), in $\{\texttt{0}, \texttt{1}\}$ for bits(l), in $[0, 1]$ for reals(l), and $\{1, \dots, |\mathcal{R}_{n_i}|\}$ (with the appropriate value for i) for structured(l, n_{rec}).

Concerning the mutation operator, we use the point-mutation, that randomly changes each genotype element to another value in the proper domain with p_{mut} probability, for ints(l, b), bits(l), and structured(l, n_{rec}). For reals(l) we use the Gaussian mutation with σ_{mut}.

Finally, concerning the crossover operator, we use the uniform crossover, that takes each element in the child genotype from one of the parents with equal probability, for all the representations.

3 Experiments and results

We performed some experiments to: (a) compare the different variants of the development algorithm (i.e., sorting criterion and overwriting flag), (b) compare the different representations, (c) verify if our approach actually allows to evolve polyominoes towards a predefined target shape while adhering to the given grammar. Concerning the representations, we experimented with bits(l), ints($l, 4$), ints($l, 16$), reals(l), and structured($l, 2$), with different values for the genotype length l.

When comparing variants and representation, we focused on analyzing quantitatively some properties of the representation, since they allow to characterize how the search process will work [19, 30]. Namely, we consider the following quantitative properties, which we measured experimentally:

Validity It measures the degree to which a genotype is mapped to a valid phenotype. Given a set G of genotypes, we obtained the corresponding bag P of phenotypes by applying our development algorithm and then we computed the validity as $\frac{1}{|G|}|\{p \in P : p \neq \varnothing\}|$.

Uniqueness It measures the degree to which different genotypes are mapped to different phenotypes. Given a set G of genotypes and the corresponding bag P of phenotypes, we computed the uniqueness as $\frac{|G|}{|P'|}$, with P' being the set of elements of P different than $\varnothing$, i.e., the valid phenotypes. Note that P may contain duplicates, while P' does not, being a set.

Locality It measures the degree to which similar genotypes are mapped to similar phenotypes. Given a sequence G of unique genotypes, the corresponding sequence P of phenotypes, and two distances d_G and d_P defined for genotypes and phenotypes, we computed the distance matrices $\boldsymbol{D}_G$ and $\boldsymbol{D}_P$ containing the distances between all pairs of elements of the two sequences and then we computed the locality as the Pearson correlation between the corresponding elements of the matrices. As d_P we used the Hamming distance of pair of polyominoes after having translated them in order to have coincident centers of mass. As d_G we used Hamming distance for bits(l), ints(l, b), and structured$(l, 2)$, Euclidean distance for reals(l).

For all the properties, the greater, the better.

We performed our experiments with five PoCFGs, shown in Figure 2. They differ in the number $|\mathcal{R}|$ of rules, the number $|T|$ of terminals, and the number $|N|$ of non-terminals.

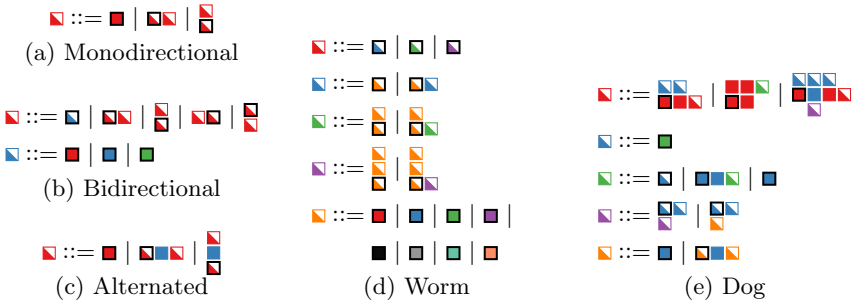


Fig. 2: The PoCFGs considered in the experiments. Half colored squares ◪ represent non-terminal symbols; fully colored squares ■ represent terminal symbols; on the right-hand-side, a thick black border ■ denotes the reference cell in a referenced polyomino. E.g., for the Dog grammar, $N = \{$◪, ◪, ◪, ◪, ◪$\}$, $T = \{$■, ■, ■$\}$, $n_1 =$ ◪, and there are $|\mathcal{R}| = 11$ production rules.

3.1 Comparison of development variants

For comparing the six variants of our development algorithm obtained by combining the three sorting criteria and the two values for the overwriting flags, we used the bits(l) representation, with $l \in \{10, 15, \ldots, 245, 250\}$. We performed similar experiments also for the other representations, observing qualitatively similar findings. For measuring the properties, for each l value we generated 5000 genotypes (and the corresponding phenotypes) for validity and uniqueness and 1000 genotypes for the locality. Figure 3 presents the results of this experiment.

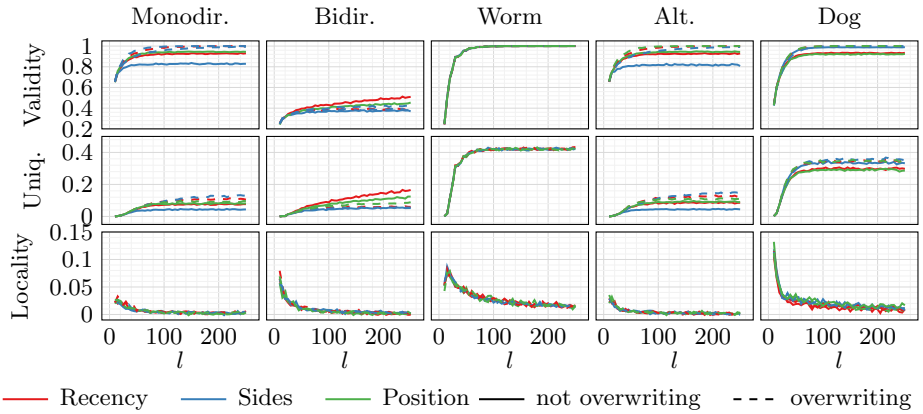


Fig. 3: Representation properties (rows of plots) for the six variants of the development algorithm (line colors and types) measured on the five grammars (columns of plots) with the bits(l) representation.

We first observe that differences in terms of properties are more apparent between PoCFGs (columns of plots) than between variants of the algorithm (line colors). That is, the grammar plays a key role in determining the properties of the representation. This finding is consistent with the literature of G3P algorithms, which has shown that grammar design can greatly impact the behavior of the algorithm [10, 17, 24]. On the other hand, it shows that our development algorithm is robust with respect to its parameters.

By looking at the plots related to validity (first row), it is possible to see that overwriting results, in general, in a larger number of valid polyominoes. With all the grammars, except for the Bidirectional, the validity reaches its maximum for most of the combinations with overwriting and a large enough l. The reason why Bidirectional leads to lower validity, might be related to the fact that there is a higher chance of selecting a non-terminal than a terminal, when comparing to the other grammars.

Concerning uniqueness, Figure 3 suggests that the Sides criterion tends to result in lower uniqueness and Recency in greater uniqueness. No clear and general distinctions can be made between the variants with and without overwriting.

Finally, concerning the locality, the results suggest that there are no differences among the variants. The main role is played by l: the longer the genotype, the lower the locality. This finding can be explained by the fact that long genotypes might not be used completely in the mapping process: the differences in unused parts of two genotypes would not be reflected in the corresponding phenotypes. This interplay between locality and actual usage of the genotype has already been observed in previous works and can be spotted with the help of visualization tools [23].

Based on the results of this experiment, we decided to use the Recency criterion without overwriting. Specifically, we selected the latter parameter value due to its closer alignment with the function of a grammar, namely, describing structural constraints for polyominoes.

3.2 Comparison of representations

We compared the five representations with the same procedure of the previous experiment (and with the Recency criterion without overwriting). The results are depicted in Figure 4.

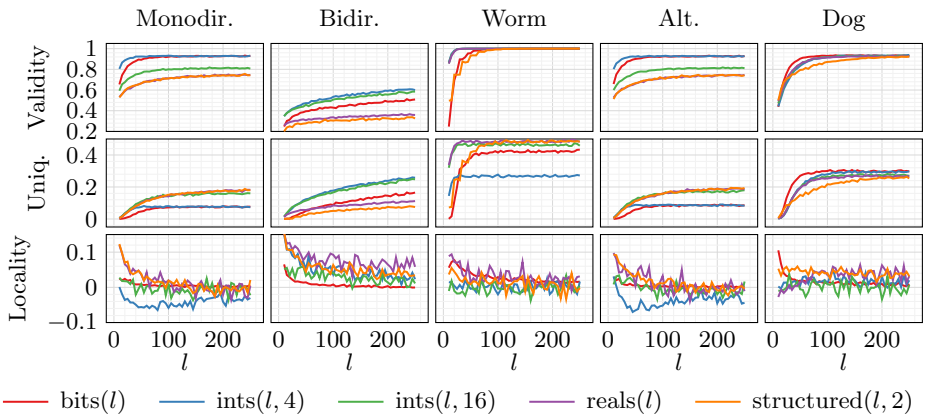


Fig. 4: Representation properties (rows of plots) for the five representations (line color) measured on the five grammars (columns of plots) with the development algorithm based on Recency without overwriting.

As in the previous analysis, the results shows that the grammar and the genotype length l impact more on the representation properties than the representation itself. However, representations differ more than development algorithm variants.

The bits(l) and ints($l, 4$) representations are in general better in terms of validity. structured($l, 2$) generates more invalid polyominoes than the other representations, likely because some portions of the genotype are not long enough—related to the n_{rec} parameter. However, larger validity does not always mean

more unique phenotypes: in all the grammars except the Bidirectional and the Dog grammar, the structured$(l, 2)$ representation presents higher uniqueness.

Concerning locality, structured$(l, 2)$ and reals(l) score, in general, better. All representations present a similar trend, except bits(l), which presents a smoother line (with, however, low locality).

Although in EAs a higher locality is important, we chose to perform the subsequent experiments with the bits(l) representation, as it is the simplest one and the most similar to the original GE.

3.3 Evolution of polyominoes

The last analysis consists in the evolution of polyominoes, as we wanted to show that the algorithm proposed can be used inside an EA to solve optimization problems. We evolved the polyominoes using the EA described in Section 2.4 with the following parameters: $n_{\text{pop}} = 100$, $n_{\text{gen}} = 200$, $r_{\text{x-over}} = 0.8$, $n_{\text{tour}} = 3$, $p_{\text{mut}} = 0.01$, the latter being the only representation-specific parameter. We employed the bits(500) representation with the Recency criterion and no overwriting. We used JGEA [22] for the experiments.

We built an optimization problem where the goal is to evolve a target polyomino $p^\star$. We used, as fitness function, the average of the Hamming distance of the evaluated polyomino p to the target $p^\star$ and the same distance computed without considering labels; in both cases, we translated the polyominoes in order to have their centers of mass to coincide. We employed this function, namely, also the part disregarding the labels, to facilitate the evolution of the correct shape.

We considered five target polyominoes, shown in Figure 5, and used each of the five PoCFGs on each target polyomino. We purposely chose target polyominoes which match very differently the five PoCFGs. Note that the Dog shape is not perfectly achievable with the Dog grammar, due to the misplaced rightmost foot.

(a) Chess (b) Circle (c) Worm-1 (d) Worm-2 (e) Dog

Fig. 5: The five target polyominoes.

For each of the 25 combinations of grammar and target, we performed 50 evolutionary runs by varying the random seed. Figure 6 presents the results of these experiments: it shows the fitness of the best polyomino during the evolution and its size, i.e., number of cells.

By looking at the results, it is possible to see that the EA successfully identified the optimal solutions for the Worm-1 and Worm-2 by employing the Worm grammar. Similarly, the Dog grammar greatly outperformed the other grammars in solving the Dog problem. This observation highlights the significance of

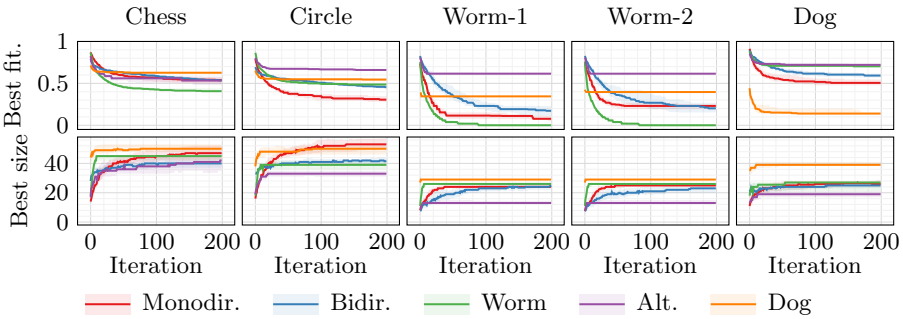


Fig. 6: Fitness and size (row of plots) of the best polyomino during the evolution for the five problems (column of plots) using the five grammars (color line) with the development algorithm based on Recency without overwriting and the bits(500) representation. The shaded area corresponds to the interquartile range, the line to the median across 50 runs.

a well-designed grammar, not only to specify structural constraints, but also to incorporate domain-specific knowledge about the problem.

When employing a grammar not specifically designed for the problem at hand, such as the Alternated grammar for Circle, Worm-1, Worm-2, and Dog problems, the EA seems to become trapped in local minima after a few iterations. This is indicated by the size of the fittest individual, which remains unchanged, highlighting the EA difficulty to explore the solution space effectively under these circumstances.

All to all, these experiments show that it is possible to evolve a polyomino towards a specific target while keeping it satisfying some predefined constraints, encoded in a user defined PoCFG.

4 Conclusions

This work introduces the concept of PoCFGs and a novel approach to generate polyominoes that meet predetermined requirements, defined by a PoCFG. The experimental results align with existing literature of G3P algorithms, highlighting the importance of grammar design [10, 17, 24], as differences in terms of representation properties are more apparent between PoCFG, than between variants of the algorithm. Additionally, we show the adaptability and potential of our approach by integrating the algorithm within EA to evolve polyominoes towards a specific target while satisfying some predefined constraints encoded in a designed PoCFG.

Future work aims to explore the applicability of the algorithm by evolving polyominoes in more complex problems, such as the generation and evolution of modular robots [28], maps for games [11], or DNA shapes [34].

Acknowledgements

This research is the result of the collaboration with the Department of Engineering and Architecture of the University of Trieste, Italy; supported by the 2023 SPECIES scholarship. The first author is funded by FCT - Foundation for Science and Technology, under the grant 2022.10174.BD. This work was supported by the Portuguese Recovery and Resilience Plan (PRR) through project C645008882-00000055, Center for Responsible AI, by the FCT, I.P./MCTES through national funds (PIDDAC), by Project No. 7059 - Neuraspace - AI fights Space Debris, reference C644877546-00000020, supported by the RRP - Recovery and Resilience Plan and the European Next Generation EU Funds, following Notice No. 02/C05-i01/2022, Component 5 - Capitalization and Business Innovation - Mobilizing Agendas for Business Innovation, and within the scope of CISUC R&D Unit - UIDB/00326/2020.

References

[1] Barequet, G., Golomb, S.W., Klarner, D.A.: Polyominoes. In: Handbook of Discrete and Computational Geometry, pp. 359–380, Chapman and Hall/CRC (2017)
[2] Bartoli, A., Castelli, M., Medvet, E.: Weighted hierarchical grammatical evolution. IEEE transactions on cybernetics **50**(2), 476–488 (2018)
[3] Conway, A.: Enumerating 2d percolation series by the finite-lattice method: Theory. Journal of Physics A: Mathematical and General **28**(2), 335 (1995)
[4] Fernau, H., Schmid, M.L., Subramanian, K.G.: Two-dimensional pattern languages. In: Workshop on Non-Classical Models for Automata and Applications (2017)
[5] Fukuda, H., Kanomata, C., Mutoh, N., Nakamura, G., Schattschneider, D.: Polyominoes and polyiamonds as fundamental domains of isohedral tilings with rotational symmetry. Symmetry **3**(4), 828–851 (2011), ISSN 2073-8994
[6] Giammarresi, D., Restivo, A.: Two-Dimensional Languages, pp. 215–267. Springer Berlin Heidelberg, Berlin, Heidelberg (1997), ISBN 978-3-642-59126-6
[7] Golomb, S.W., Klarner, D.A.: Polyominoes. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry, Second Edition, pp. 331–352, Chapman and Hall/CRC (2004)
[8] Grimmett, G.: What is Percolation?, pp. 1–31. Springer Berlin Heidelberg, Berlin, Heidelberg (1999), ISBN 978-3-662-03981-6
[9] Hansen, N., Arnold, D.V., Auger, A.: Evolution strategies. Springer handbook of computational intelligence pp. 871–898 (2015)
[10] Harper, R.: GE, explosive grammars and the lasting legacy of bad initialisation. In: IEEE Congress on Evolutionary Computation, IEEE (Jul 2010)
[11] Johnson, L., Yannakakis, G.N., Togelius, J.: Cellular automata for real-time generation of infinite cave levels. In: Proceedings of the 2010 Workshop on Procedural Content Generation in Games, pp. 1–4 (2010)

[12] Knight, T., Stiny, G.: Making grammars: from computing with shapes to computing with things. Design Studies **41**, 8–28 (2015)
[13] Knuth, D.E.: Dancing links. arXiv preprint cs/0011047 (2000)
[14] Křivka, Z., Martín-Vide, C., Meduna, A., Subramanian, K.G.: A variant of pure two-dimensional context-free grammars generating picture languages. In: Barneva, R.P., Brimkov, V.E., Šlapal, J. (eds.) Combinatorial Image Analysis, pp. 123–133, Springer International Publishing, Cham (2014), ISBN 978-3-319-07148-0
[15] Lavirotte, S., Pottier, L.: Optical formula recognition. In: Proceedings of the Fourth International Conference on Document Analysis and Recognition, vol. 1, pp. 357–361, IEEE (1997)
[16] Lourenço, N., Pereira, F.B., Costa, E.: Unveiling the properties of structured grammatical evolution. Genetic Programming and Evolvable Machines **17**, 251–289 (2016)
[17] Manzoni, L., Bartoli, A., Castelli, M., Gonçalves, I., Medvet, E.: Specializing context-free grammars with a (1+ 1)-ea. IEEE Transactions on Evolutionary Computation **24**(5), 960–973 (2020)
[18] Matz, O.: Regular expressions and context-free grammars for picture languages. In: Reischuk, R., Morvan, M. (eds.) STACS 97, pp. 283–294, Springer Berlin Heidelberg, Berlin, Heidelberg (1997), ISBN 978-3-540-68342-1
[19] Medvet, E.: A comparative analysis of dynamic locality and redundancy in grammatical evolution. In: European Conference on Genetic Programming, pp. 326–342, Springer (2017)
[20] Medvet, E., Bartoli, A., De Lorenzo, A., Tarlao, F.: Gomge: Gene-pool optimal mixing on grammatical evolution. In: Parallel Problem Solving from Nature–PPSN XV: 15th International Conference, Coimbra, Portugal, September 8–12, 2018, Proceedings, Part I 15, pp. 223–235, Springer (2018)
[21] Medvet, E., Nadizar, G.: GP for Continuous Control: Teacher or Learner? The Case of Simulated Modular Soft Robots. Genetic Programming Theory and Practice XX (2023)
[22] Medvet, E., Nadizar, G., Manzoni, L.: JGEA: a modular java framework for experimenting with evolutionary computation. In: Proceedings of the genetic and evolutionary computation conference companion, pp. 2009–2018 (2022)
[23] Medvet, E., Virgolin, M., Castelli, M., Bosman, P.A., Gonçalves, I., Tušar, T.: Unveiling evolutionary algorithm representation with du maps. Genetic Programming and Evolvable Machines **19**, 351–389 (2018)
[24] Nicolau, M., Agapitos, A.: Understanding grammatical evolution: Grammar design. In: Handbook of Grammatical Evolution, pp. 23–53, Springer International Publishing (2018)
[25] Noya, E., Benedí, J.M., Sánchez, J.A., Anitei, D.: Discriminative learning of two-dimensional probabilistic context-free grammars for mathematical expression recognition and retrieval. In: Pinho, A.J., Georgieva, P., Teixeira, L.F., Sánchez, J.A. (eds.) Pattern Recognition and Image Analysis, pp. 333–347, Springer International Publishing, Cham (2022)

[26] Ong, H.S., Syafiq-Rahim, M., Kasim, N.H.A., Firdaus-Raih, M., Ramlan, E.I.: Self-assembly programming of dna polyominoes. Journal of Biotechnology **236**, 141–151 (2016), ISSN 0168-1656
[27] Ota, P.A.: Mosaic grammars. Pattern recognition **7**(1-2), 61–65 (1975)
[28] Pigozzi, F., Medvet, E., Bartoli, A., Rochelli, M.: Factors impacting diversity and effectiveness of evolved modular robots. ACM Transactions on Evolutionary Learning **3**(1), 1–33 (2023)
[29] Prusa, D., Hlavá, V.: 2d context-free grammars: Mathematical formulae recognition. In: Prague Stringology Conference (2006)
[30] Rothlauf, F., Goldberg, D.E.: Redundant representations in evolutionary computation. Evolutionary Computation **11**(4), 381–415 (2003)
[31] Ryan, C., Collins, J.J., Neill, M.O.: Grammatical evolution: Evolving programs for an arbitrary language. In: Genetic Programming: First European Workshop, EuroGP'98 Paris, France, April 14–15, 1998 Proceedings 1, pp. 83–96, Springer (1998)
[32] Sakai, I.: Syntax in universal translation. In: Proceedings of the International Conference on Machine Translation and Applied Language Analysis (1961)
[33] Salimans, T., Ho, J., Chen, X., Sidor, S., Sutskever, I.: Evolution strategies as a scalable alternative to reinforcement learning. arXiv preprint arXiv:1703.03864 (2017)
[34] San Ong, H., Syafiq-Rahim, M., Kasim, N.H.A., Firdaus-Raih, M., Ramlan, E.I.: Self-assembly programming of dna polyominoes. Journal of Biotechnology **236**, 141–151 (2016)
[35] Subramanian, K., Ali, R.M., Geethalakshmi, M., Nagar, A.K.: Pure 2d picture grammars and languages. Discrete Applied Mathematics **157**(16), 3401–3411 (2009)
[36] Thierens, D., Bosman, P.A.: Optimal mixing evolutionary algorithms. In: Proceedings of the 13th annual conference on Genetic and evolutionary computation, pp. 617–624 (2011)
[37] Whittington, S.G., Soteros, C.E.: Lattice animals: Rigorous results and wild guesses (1990)
[38] Winslow, A.: Staged self-assembly and polyomino context-free grammars. Natural Computing **14**(2), 293–302 (2015)