Exploring Grammar-Guided Design and Evolution of Polyominoes with Modular Soft Robots

Jessica Mégane[1*], Eric Medvet[2], Nuno Lourenço[1], Penousal Machado[1]

[1]University of Coimbra, CISUC/LASI – Centre for Informatics and Systems of the University of Coimbra, Department of Informatics Engineering, Coimbra, Portugal.
[2]Department of Engineering and Architecture, University of Trieste, Trieste, Italy.

*Corresponding author(s). E-mail(s): jessicac@dei.uc.pt;
Contributing authors: emedvet@units.it; naml@dei.uc.pt;
machado@dei.uc.pt;

Abstract

Languages for describing two-dimensional (2-D) structures have become powerful tools across multiple fields, including pattern recognition, image processing, and the modeling of physical and chemical phenomena. One of such structures is labeled polyominoes, *i.e.*, geometric shapes formed by connected unit squares arranged on a 2-D grid. In previous work, we introduced: (a) a novel grammar-based approach for defining sets of labeled polyominoes that satisfy predefined requirements, and (b) an algorithm to develop labeled polyominoes following the rules of the proposed grammar. We demonstrated that these two components enable optimization within the space of labeled polyominoes, similarly to how grammatical evolution and its extensions operate in string-based search spaces. In this work, we extend our previous approach to a new domain: the evolution of modular soft robots, namely, voxel-based soft robots (VSRs). We evolve VSRs for the task of energy-efficient locomotion, while constraining their physical structure to adhere to a given grammar. We show that the evolved robots successfully perform their assigned tasks and do have the required structure. These results highlight the potential of integrating domain knowledge through grammars to guide the evolutionary design of complex structure as modular soft robots.

Keywords: polyomino, grammar, representation, 2-D patterns, voxel-based soft robots

1 Introduction

Two-dimensional (2-D) languages have emerged as powerful tools across various fields, originally motivated by challenges in pattern recognition and image processing [21, 26, 39]. These languages generate 2-D objects, ranging from simple rectangles to more complex shapes such as decagons. Among these, polyominoes have attracted significant interest due to their unique properties [4, 24, 38] and diverse applications [29, 40, 62, 63, 68, 70, 75].

A polyomino [27] is defined as a finite connected set of unit squares arranged on a 2-D lattice. In the physical and chemical sciences, these structures are often referred to as lattice animals [29, 63] and are commonly employed to model branched polymers, molecular structures, and percolation processes, offering insights into complex systems [11, 82]. Their mathematical properties have led to extensive study in combinatorial optimization and mathematics [4, 24, 38], but also influenced theoretical formal language research [26, 27]. A common task in many of these fields is to identify one or more polyominoes that maximize specific objectives while meeting predefined structural requirements [2]. This can be effectively addressed using formal mechanism such as grammars to describe and generate 2-D shapes [40, 62, 64, 70, 83].

Grammars formally describe a language through a set of production rules. To generate 2-D shapes, they may encode 2-D structures through one-dimensional (1-D) strings [83] or operate directly on 2-D representations [64, 70]. Grammars have been used to generate and study the properties of polyominoes [83], and their use in generating 2-D pictures [37, 39, 47, 76] is well established, with practical applications in tasks such as mathematical formula recognition [40, 62, 70].

In earlier work [55], we proposed a novel approach to generate labeled polyominoes that comply with formally defined structural requirements, where labels assign additional information to individual cells. Specifically, we: (a) introduced polyomino context-free grammar (PoCFG) as an extension of context-free grammars (CFGs), and (b) proposed a development algorithm that can be used for generating polyominoes that conform to a given PoCFG. The proposed algorithm constructs these polyomino structures with precise control of the shape of the polyomino and labeling of its cells. When integrated within an evolutionary algorithm (EA), it enables the optimization of labeled polyominoes that adhere to a given PoCFG. The approach simply requires providing a grammar and a fitness function to the selected EA; notably, it eliminates the need for user-defined variation operators that guarantee that the resulting polyominoes adhere to the PoCFG—*i.e.*, operators with the closure property. This fact greatly increases the applicability of our method, simplifying polyomino optimization, much like grammatical evolution (GE) [73] did for regular languages.

Since our algorithm is greatly agnostic with respect to the genotypic representation used by the EA, we compared different representations based on validity, redundancy, and locality. To showcase its effectiveness, we evolved polyominoes in a few case studies, where the goal is to evolve a polyomino as similar as possible to a pre-defined target, while adhering to a grammar. The experimental results confirm that evolutionary optimization does work, generating polyominoes that progressively improves similarity to the target shape, while all generated polyominoes satisfy the user-defined grammatical constraints.

In this work, we extend our previous research presented in [55] in three ways.

(1) We analyze more in detail the relevant literature concerning the optimization of 2-D structures.
(2) We experimentally validate the generality of our approach by showing that our algorithm can be indeed paired with different EAs and genotype spaces: this way, one can accommodate different needs as, e.g., solving multi-objective optimization problems involving polyominoes or searching the space of polyominoes "in a quality-diversity (QD) way".
(3) We consider a new case study where we apply our development algorithm to the evolution of a kind of modular soft robot, voxel-based soft robots (VSRs), whose physical structure can be described with a labeled polyomino. We consider the non-trivial task of energy-efficient locomotion, a bi-objective optimization problem where we look for robots which run fast and energy-efficiently at the same time. We compare two cases: one where we constrain the search to VSRs whose physical structure adheres to a given grammar and one where their structure is free (but limited in size).

We show that through our approach for describing sets of labeled polyominoes with a PoCFG and developing them in the context of an evolutionary optimization one can conveniently introduce some domain knowledge (and structural constraints) in complex problems while still using general-purpose EAs.

The remainder of the paper is structured as follows. In Section 2, we present the related work on polyominoes and their applications. In Section 3, we introduce the formal definitions of polyomino and PoCFG and we describe in detail the development algorithm. In Section 4, we detail the experimental setup and discuss the results, including the novel case study about the evolution of VSRs. Finally, in Section 5 we summarize the main findings and suggest directions for future research.

2 Background and related works

2-D formal languages have been widely explored as a means to describe structured patterns across various domains. In this paper, we focus on polyominoes, which are geometric figures composed of one or more unit squares (or cells) joined edge-to-edge, and we discuss their characteristics and applications.

Polyominoes can be categorized based on properties such as compactness, symmetry, and convexity. Compactness refers, intuitively, to the presence of holes in the polyomino. Symmetry refers to invariance under transformations like rotations and reflections. Based on equivalence under these transformations, polyominoes are classified as: fixed (all orientations/positions are distinct), one-sided (rotations are equivalent, reflections are not), and free (rotations and reflections are equivalent). Convexity is defined on the grid: a polyomino is convex if it is row-convex (each row forms one contiguous cell block) and column-convex (each column forms one contiguous

block), meaning the shape has no inward dents or gaps along grid lines. Convex examples include, e.g., rectangles, L-shapes, staircases, while non-convex shapes contain at least one row or column with separated cell segments.

The study of polyominoes spans a wide range of problems and domains, bridging mathematics, computer science, and physics. Many fundamental questions remain unsolved, making polyominoes interesting for researchers. One classic challenge is counting the number of distinct polyominoes which meet some given criteria [4, 13]. This well-known combinatorial problem lacks a general formula for the number of polyominoes of a given size in most cases, despite decades of work. Recent advances have extended the maximum size for which the total count of distinct shapes is known [3, 46].

Symmetry in polyominoes plays a central role in computational geometry, particularly in classifying tiling patterns and analyzing their properties. Tiling involves covering the plane with a repeating pattern of a single shape that fits with itself in multiple orientations [24]. While some heuristics exist to determine whether a given polyomino can tile the plane, no general algorithm efficiently solves this problem in all cases. Beyond theoretical interest, tiling problems have practical implications in materials science and physics, where polyomino tilings model atomic arrangements in quasicrystals and other structures. Additionally, tiling with polyominoes inspires numerous challenges and popular games [45].

While polyominoes continue to be studied for their intrinsic mathematical properties, their structured nature also makes them valuable models for solving complex problems across diverse fields. In some contexts, polyominoes are also referred to as lattice animals [82].

One prominent application is in percolation theory [11, 29], which examines the behavior of networks as nodes or links are added. Percolation models are widely used to analyze processes such as fluid movement through porous materials [35], the spread of diseases [7], and information diffusion in networks [33]. These models often represent the system as a lattice (e.g., a grid) where each site or bond is randomly occupied with a certain probability. As the probability increases, clusters of connected occupied sites form, frequently resembling polyomino-like structures, which are critical for understanding phase transitions and connectivity in such systems.

In polymer chemistry, branched polymers (molecules with structures that resemble trees) can be mapped to polyominoes on a lattice. Each polymer molecule structure corresponds to a cluster of connected sites, which is essentially a polyomino shape. This polyomino representation enables the application of statistical mechanics on a grid to study polymer behavior [80, 81].

Polyominoes have proven useful in modeling self-assembly processes, where components spontaneously organize into ordered structures through local interactions, without external control. These models have been applied in different domains, including the assembly of DNA tiles [25, 63], the self-folding of three-dimensional (3-D) shells [69], and modular robotics [72]. In robotics, polyomino (2-D) and polycube (3-D) configurations enable adaptable, reconfigurable designs [9, 34, 41, 44, 72], supporting tasks such as locomotion, manipulation, and structural adaptation. In theoretical computer science, self-assembly systems based on polyomino tiles have been shown to achieve computational universality [18].

From combinatorial enumeration to self-assembly and robotics, polyominoes provide a powerful framework for studying structure, optimization, and emergent complexity. To effectively represent and exploit these shapes, linguistic methods have extended formal languages beyond 1-D strings to 2-D arrays [21, 26]. These approaches enable the systematical generation and analysis of not only individual polyominoes but also of entire sets, following specific rules.

Several computational models have been proposed to construct polyominoes under different assumptions. For instance, a polyomino tile assembly model has been explored in the context of self-assembly [18], where polyomino shapes emerge from the bonding of smaller tiles. Another approach, entitled Polyomino Development Algorithm, encodes an n-omino (polyominoes with n cells) as an array of integers and uses an EA to grow the polyomino shape over time [2]. Additionally, an algorithm has been proposed to generate sets of polyominoes according to their site-perimeter, which is the number of cells outside a polyomino that touch its border, within the application for the two-player game Gomino [22]. Each of these methods offers a unique perspective on polyomino construction, and together they provide valuable tools for exploring both the theory and practical design of polyomino structures.

Among modeling tools, cellular automata (CA) and grammars stand out for their ability to incorporate structural constraints while supporting flexible shape generation. Both have been successfully adapted to study polyomino properties and their dynamics. CA [65] operate on an infinite or bounded grid of cells, each having a finite number of states, and have been widely employed to grow 2-D shapes [56], including polyominoes. As a dynamic representation, CA use local transformation rules to model the growth of a polyomino over time, cell by cell. In the literature, CA have been applied to investigate various properties of polyominoes, such as tiling patterns [1, 4] and shape convexity [28], by observing how clusters of cells evolve under different sets of rules.

Grammars, commonly used for 2-D pattern languages [26, 70], offer an alternative mechanism to generate and analyze polyomino shapes, grounded in rule-based derivation. A grammar defines how shapes are constructed by recursively applying production rules, which can be designed to integrate domain-specific knowledge and/or constraints, for example the presence of holes or convexity. Grammar-based methods have been applied to describe and enumerate various classes of polyominoes, especially convex polyominoes [16, 17, 23], and to support enumeration tasks using attribute grammars [14]. Beyond polyomino enumeration, spatial grammars have also been used in generative design of 3-D soft robots [15], which effectively take the form of 3-D polyominoes. In this approach, grammar rules directly encode design requirements and constraints, instead of relying on the objective functions. This approach has demonstrated the ability to generate both expected and novel designs, with all outcomes constrained by the grammar. Different approaches were tested and show that single applied grammar rules were able to increase the objective function, highlighting their potential. Grammars have also been proposed to represent and generate polymers [30], successfully reproducing large datasets compiled from the literature. This representation has proven useful for reverse engineering of polymer design and production.

Fig. 1: Example of polyominoes with labels in $A = \{$■, ■, ■, ■, ■, ■, ■$\}$.

Moreover, it offers a blueprint for applications in other fields, such as chemical model design, molecular discovery, and property optimization.

The present study builds on a previous work in which we proposed a grammar-based approach for describing sets of labeled polyominoes, along with an algorithm to develop shapes that respect the grammar constraints [55]. We also demonstrated how these components can be used in optimization tasks. The results showed that grammars can effectively enforce hard constraints within the polyomino search space while supporting evolutionary search. This framework is generalizable: to apply it to a specific problem, one only needs to design a grammar that encodes the desired restrictions. If optimization is the goal, then, like in any usage of an EA, an appropriate objective function must also be defined.

3 Grammar-based polyominoes and their evolution

3.1 Labeled polyomino

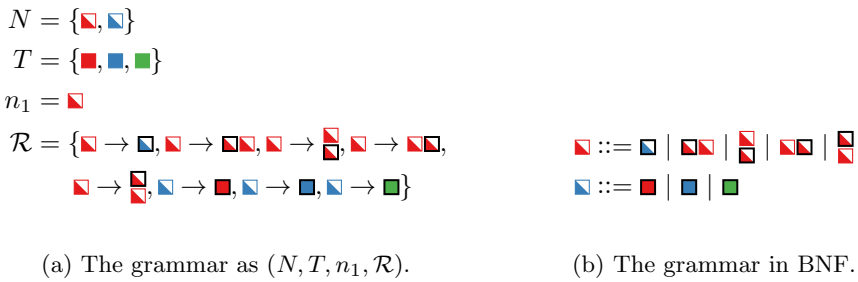
A *polyomino* is a 2-D geometric figure composed by one or more squares (or *cells*) joined along their edges. A *labeled polyomino* over an alphabet A is a polyomino where each cell is assigned a symbol (or *label*) $a \in A$. For brevity, we will refer to labeled polyominoes simply as polyominoes. Let $\mathcal{P}_A$ denote the set of all the polyominoes defined over A. Figure 1 shows examples of polyominoes defined over an alphabet of seven symbols, encoded as colors for easing the visualization.

Given a polyomino p, we assign coordinates $(x_0, y_0) \in \mathbb{Z}^2$ to one of its cells. The label of a cell displaced by $x - x_0$ cells along the x-axis and $y - y_0$ cells along the y-axis relative to (x_0, y_0) is denoted as $p_{x,y} =\in A$. If no cell exists at (x, y), we define $p_{x,y} = \varnothing$.

A *referenced polyomino* is a polyomino in which one cell is designated as the *reference cell*. By convention, this reference cell is assigned the coordinate $(0, 0)$.

3.2 Polyomino context-free grammar (PoCFG)

A PoCFG $\mathcal{G}$ is defined as 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*. Each production rule consists of a non-terminal symbol (left-hand side) and a referenced polyomino defined over the alphabet $N \cup T$ (the right-hand-side).



(a) The grammar as $(N, T, n_1, \mathcal{R})$. (b) The grammar in BNF.

Fig. 2: Example of a PoCFG with $|\mathcal{R}| = 8$ production rules, two non-terminal, and three terminal symbols—throughout the paper, we will show non-terminal symbols as half colored squares. The grammar is shown as a tuple (left) and in BNF (right). In the BNF, the vertical bar | separates alternative productions for a non-terminal. The eight rules include five expansions for ◹, one rewriting into ◺, and four adding ◹ in relative positions (right, top, left, bottom), and three expansions for ◺, each producing a terminal symbol (■, ■, ■). On the right-hand-side of each rule, the thick black border ■ marks the reference cell within a referenced polyomino.

We represent a PoCFG using a compact notation resembling Backus-Naur form (BNF), similarly to the case of CFGs for strings, where production rules are grouped by non-terminal symbol. The first non-terminal corresponds to the starting symbol n_1. Within groups, the vertical bar | separates alternative production rules. Figure 2 presents an example of PoCFG in BNF, with non-terminals $N = \{$◹, ◺$\}$, terminals $T = \{$■, ■, ■$\}$, starting symbol $n_1 =$ ◹, and $|\mathcal{R}| = 8$ total rules. Throughout the paper, we will show non-terminal symbols as half colored squares.

The non-terminal symbol ◹ has five alternatives. The first rule expands into the non-terminal symbol ◺. The remaining four attach a non-terminal symbol ◹ to the current symbol being expanded in different relative positions, right, top, left, or bottom, respectively. The non-terminal ◺ has three production rules, each expanding into a terminal symbol (■, ■, or ■). Every right-hand side polyomino includes a marked reference cell (thick black border), which defines the alignment point for insertion during derivation.

A PoCFG $\mathcal{G} = (N, T, n_1, \mathcal{R})$ provides a compact way to define a (possibly infinite) set of polyominoes over T, denoted by $\mathcal{P}_\mathcal{G} \subseteq \mathcal{P}_T$—Figure 3 shows a few polyominoes belonging to the set defined by the grammar shown in Figure 2. The next section describes a constructive process to obtain a polyomino $p \in \mathcal{P}_\mathcal{G}$.

Deciding whether a given polyomino p belongs to $\mathcal{P}_\mathcal{G}$ is beyond the scope of this paper. However, for CFGs that satisfy certain conditions, this problem is solvable for the case of strings [74].

3.3 Developing polyominoes given a PoCFG

In our previous work [55] we introduced a *development algorithm* for obtaining a polyomino $p \in \mathcal{P}_\mathcal{G}$ from a PoCFG $\mathcal{G}$. The algorithm iteratively expands a polyomino

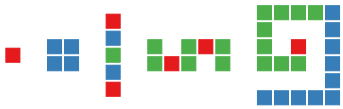


Fig. 3: Example of polyominoes belonging to $\mathcal{P}_\mathcal{G}$ with the PoCFG $\mathcal{G}$ of Figure 2.

by adding or modifying cells according to the production rules of $\mathcal{G}$, starting from the single cell polyomino given by the axiom. This process resembles a developmental model, where structures emerge step by step.

3.3.1 Design principles

The development algorithm is designed for use within an evolutionary optimization process over $\mathcal{P}_\mathcal{G}$. It takes as input the *genotype* $g \in G$, where G denotes the genotype space, which guides the selection of production rules to apply. The development algorithm thus acts as a mapping from a genotype g to a polyomino p.

To ensure flexibility, the algorithm is agnostic to the genotype type, allowing it to work, in principle, with any genotype space G. We achieved this by making the algorithm modular. Specifically, we decoupled the components responsible for rule selection, rule expansion, and identification of suitable rules—only the first component actually deals with the genotype g. This modular design allows us to easily experiment with different genotype representations. In Section 4.2, we compare various realizations of the development algorithm using different genotype structures, including bit-strings, integer-strings, and more complex data structures.

3.3.2 Development algorithm

Algorithm 1 outlines our development algorithm using pseudocode. The algorithm takes as input a genotype $g \in G$, a PoCFG $\mathcal{G}$, and two parameters: a sorting criterion c and an overwriting flag o. It returns either a polyomino $p \in \mathcal{P}_\mathcal{G}$ or $\varnothing$ if no valid polyomino can be developed with the given inputs.

The algorithm proceeds as follows. First, it sets (line 2) p as a single-cell polyomino, where the only cell is labeled with the axiom n_1. Then, it iteratively modifies p through these steps:

(i) it finds (line 5) all cells in p labeled with a non-terminal symbol in N, *i.e.*, cells that can be replaced using a production rule in $\mathcal{R}$;
(ii) it chooses (line 9) one of the identified cells $(x^\star, y^\star)$ as the target for replacement, according to the sorting criterion c;
(iii) it selects a suitable production rule for the target cell (line 11), based on the genotype g and the current state s (initialized as $\varnothing$; see Section 3.3.3).
(iv) finally, if possible, it modifies p by replacing the target cell with the referenced polyomino p', aligning its reference cell at $(x^\star, y^\star)$.

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*) ← SELECTNONTERMINAL(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) ← CHOOSEREPLACEMENT(R_n, g, s)       ▷ choose one replacing ref. pol.
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
```

The process repeats until either (a) no non-terminal cells remain in p, or (b) one of the steps cannot be performed (discussed below).

Since multiple non-terminal cells may exist in p at step (ii) above (SELECTNONTERMINAL() procedure in Algorithm 1), determining which one to expand is crucial. The choice can impact on whether subsequent steps succeed, as some production rules may become inapplicable, depending on the selected cell. We treat this as a sorting problem and explore three sorting criteria for the selection of a non-terminal cell:

Position criterion: Select the non-terminal cell in p with the lowest y-coordinate; in case of tie, choose the one with the lowest x-coordinate.

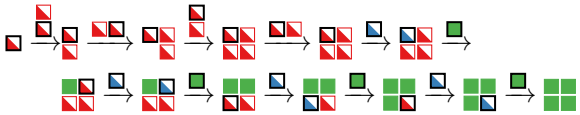
Recency criterion: Select the non-terminal cell most recently added to p (*i.e.*, in the most recent iteration of the algorithm); in case of tie, apply the Position criterion.

Free sides criterion: Select the cell with the greatest number of free sides (*i.e.*, sides without adjacent cells); in case of tie, apply the Position criterion.

In Algorithm 1, the parameter c determines which sorting criterion SELECTNONTERMINAL() applies. We compare different choices experimentally in Section 4.1.

Once a target cell is selected, the algorithm must determine which production rule to apply (step (iii)). The function CHOOSEREPLACEMENT() handles this step, selecting

(a) With $o = \text{false}$.



(b) With $o = \text{true}$.

Fig. 4: An example of polyomino development based on the PoCFG from Figure 2, without ($o = \text{false}$, above) and with ($o = \text{true}$, below) overwriting, using the Position criterion. Polyominoes between arrows represent intermediate development stages: the thick black border denotes the cell being replaced. The referenced polyominoes above the arrows correspond to the right-hand-side of the applied production rule: here, the thick black border denotes the reference cell.

among the applicable rules based on the genotype g. We explore three alternative implementations of this function in Section 3.3.3.

No-mapping conditions

The development algorithm returns $\varnothing$ (*i.e.*, fails in mapping a genotype g to a polyomino $p \in \mathcal{P}_\mathcal{G}$) in two cases. The first occurs when no production rule can be selected for a replaceable cell because CHOOSEREPLACEMENT$(\mathcal{R}_r, g)$ returns $\varnothing$, which typically happens when g has been completely consumed (see next section). This mechanism prevents infinite execution, similar to termination strategies in other GE variants, such as structured grammatical evolution (SGE) [42] and weighted hierarchical grammatical evolution (WHGE) [5]. The second case arises when the selected cell of p at $(x^\star, y^\star)$ cannot be replaced by the referenced polyomino p' from the chosen rule—Figure 4 shows an example of such a situation. This happens if nearby non-empty cells in p would need to be replaced by corresponding cells in p', but a conflict prevents the replacement. Unlike string-based grammars, where symbols can be replaced by sequences by shifting surrounding substrings, no such flexibility exists in 2-D structures. In Algorithm 1, this condition is verified by FITS(). A variant of the algorithm allows rules to be applied regardless of conflicts, enabling overwriting when placing p' at $(x^\star, y^\star)$. The Boolean parameter o in Algorithm 1 controls this behavior, representing an overwriting flag.

3.3.3 Different types of genotype

The development algorithm is designed to support different *representations*, allowing for different genotype domains G. Our rationale is twofold. First, we aimed to demonstrate that the algorithm is general, by decoupling the choice of the production rules from the rest of the process. Second, we wanted to build on prior research and established practices in grammar-guided genetic programming (G3P), where different kinds of genotype (and different ways of using them) have been explored to enhance the general effectiveness of the evolutionary search, such as bit-strings in early GE and WHGE, as well as structured strings of integers in SGE.

We implemented four representations, each corresponding to a different implementation of the CHOOSEREPLACEMENT() function in Algorithm 1. The modular design of the algorithm allows these representations to be incorporated seamlessly. In all cases, we assume that the selection of production rules based on the genotype is stateful, meaning that repeated calls of CHOOSEREPLACEMENT() with the same g may yield different outputs. To formalize this, we include a state s as an argument for CHOOSEREPLACEMENT() and define the function to return an updated state s along with the selected reference polyomino p'. The state is initialized to an empty value $\varnothing$ at the beginning of each execution of the development algorithm, and its domain varies depending on the chosen representation. We remark that DEVELOP(), the algorithm in which CHOOSEREPLACEMENT() is used one or more times, is not stateful nor stochastic, hence different calls of DEVELOP() with the same g (and same PoCFG, c, and o) always result in the same outcome.

Figure 5 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 an integer counter, *i.e.*, $s \in S = \{1, \ldots, l\} \in \mathbb{N}$.

Given a genotype $g = (g_1, \ldots, g_l)$, a set of 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 being a referenced polyomino defined over $N \cup T$), and the state s, the function CHOOSEREPLACEMENT() operates as follows. If the state s is $\varnothing$, it is initialized to 1; otherwise, it is incremented by 1. If $s > l$, the function returns $p' = \varnothing$; otherwise, the reference polyomino is selected as $p' = p_j$, with $j = ((g_s - 1) \bmod k) + 1$. Intuitively, CHOOSEREPLACEMENT() consumes the genotype one integer at a time and selects the rule using the mod rule, as in the original GE.

This representation is denoted as $\text{ints}(l, b)$, where l is the genotype length, and b is the maximum possible value of each genotype element. That is, l and b are parameters for this parametric representation.

String of bits.

In this representation, the genotype is a binary string, $G = \{\texttt{0}, \texttt{1}\}^l$, with the state $S = \{1, \ldots, l\} \in \mathbb{N}$.

Given $g = (g_1, \ldots, g_l)$, a set of production rules $\mathcal{R}_n = (r_1, \ldots, r_k)$ for the non-terminal n, and a state s, CHOOSEREPLACEMENT() works as follows. If the state, s, is

$g = (\overbrace{2, 8, 6, 1, 6, 3, 6, 7}^{(l=8)})$

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

(a) With ints$(8, 8)$.

$g = \overbrace{0111000100\ldots11}^{(l=32)}$

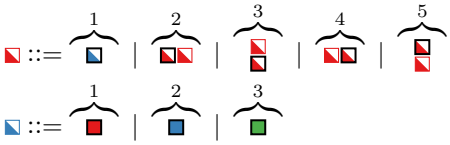
p	s	h	g'	z	k	j	p'
	1	3	011	3	5	4	
	4	3	100	5	5	1	
	7	2	01	1	3	2	
	9	3	001	1	5	2	
	12	3	000	0	5	1	
	15	2	11	3	3	1	
	17	3	101	5	5	1	
	20	2	11	3	3	2	
	22						

(b) With bits(32).

$g = (\overbrace{5, 2, 5, 1, \ldots}^{(l_1=73)}, \overbrace{3, 2, 3, \ldots}^{(l_2=9)})$

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$(82, 2)$.



(d) The grammar $\mathcal{G}$.

Fig. 5: Example of polyomino development using the grammar from Figure 2 (also shown in 5d for clarity, with the index of each rule explicitly labeled), illustrated across three representations (one table for each representation). Each row corresponds to one iteration of the algorithm using the Position sorting criterion without overwriting. The thick black border denotes in p the cell that is being replaced and in p' the reference cell.

$\varnothing$, it is set to 1; otherwise, it is incremented by $h = \lceil \log_2 |\mathcal{R}_n| \rceil$. If $s+h > l$, the function returns $p' = \varnothing$; otherwise: (i) it extracts the next h bits, $g' = (g_s, g_{s+1}, \ldots, g_{s+h})$ in g; (ii) it converts g' into an integer $z \in \{1, \ldots, 2^h\}$; (iii) it selects the reference polyomino as $p' = p_j$, with $j = (z \bmod k) + 1$.

In this case, chooseReplacement() processes h bits at a time, ensuring that h is the smallest number needed to accommodate $\mathcal{R}_n$ and chooses the rule using the mod rule on the bit-to-integer conversion of the consumed h bits. If $|\mathcal{R}_n| = 1$, no bits are consumed—which is sound, as there are no decisions to be made. This representation is denoted as bits(l).

String of reals

In this representation, the genotype consists of real-valued elements, $G = \mathbb{R}^l$, with $S = \{1, \ldots, l\} \in \mathbb{N}$.

Given $g = (g_1, \ldots, g_l)$, a set of production rules $\mathcal{R}_r$, and state s, chooseReplacement() works similarly to ints(l, b) in updating s. The reference polyomino selection proceeds as follows. If $s > l$, the function returns $p' = \varnothing$; otherwise: (i) it clamps g_s to the range $[0, 1]$ as $h = \min(1, \max(0, g_s))$, then (ii) it selects $p' = p_j$, with $j = \max(1, \lceil k g_s \rceil)$.

Intuitively, here chooseReplacement() consumes the genotype one real value at a time and chooses the rule based on the value normalized to $[0, 1]$ and mapped to a valid rule index, $\{1, \ldots, |\mathcal{R}_n|\}$.

This representation is denoted as reals(l).

Structured string of integers

In this representation, the 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 the constraint that $\sum_{j=1}^{j=m} l_j = l$, ensuring that the total genotype length remains l.

The number of genotype elements l_j assigned to each non-terminal n_j of the grammar is determined based on the total genotype length l through the following process.

(1) We start with a bag $\mathcal{N} = \{n_1\}$ containing only the axiom.
(2) We repeat for n_{rec} times this procedure: (i) for each non-terminal n in $\mathcal{N}$, we collect all rules $\mathcal{R}_n$ associated with n; (ii) we extract all referenced polyominoes appearing on the right-hand side of these rules; (iii) we add all non-terminals from these polyominoes to $\mathcal{N}$ (potentially with repetitions).
(3) Finally, we assign a portion of the genotype to each non-terminal n_j based on its frequency in $\mathcal{N}$. Namely, for n_j we set $l_j = \left\lfloor l \frac{|\{n \in \mathcal{N} : n = n_j\}|}{|\mathcal{N}|} \right\rfloor$. The values are then adjusted to ensure that $\sum_{j=1}^{j=m} l_j = l$.

The idea behind this procedure is to allocate a suitable number of genotype elements to allow "enough" productions for each given non-terminal, while keeping the genotype length fixed at l. The parameter n_{rec} influences how much more recursive non-terminals (*i.e.*, those that appear more frequently in derivations) receive larger portions of the genotype.

Given a genotype, $g = (g_{1,1}, \ldots, g_{1,l_1}, \ldots, g_{m,1}, \ldots, g_{m,l_m})$, a set of rules $\mathcal{R}_{n_i}$ for a non-terminal n_i, and a state $s = (s_1, \ldots, s_m)$ (or $s = \varnothing$), the function CHOOSEREPLACEMENT() for this representation works as follows. If $s = \varnothing$, it initializes it as $s = (1, \ldots, 1)$ (*i.e.*, a m-long vector of ones); otherwise, it updates the counter for the current non-terminal: if $s = (s_1, \ldots, s_m)$, then $s_i = s_i + 1$. If $s_i > l_i$, $p' = \varnothing$; otherwise $p' = p_j$, with $j = g_{i,s_i}$

Intuitively, here CHOOSEREPLACEMENT() selects a production rule by sequentially consuming one integer at a time from the genotype portion corresponding to the current non-terminal being replaced. Note that no modulo operation here is required, as the domain of each genotype segment exactly matches the number of production rules for its corresponding non-terminal.

We denote this representation, which resembles that of SGE [42], as structured(l, n_{rec}).

3.4 Evolution of polyominoes

Having defined how to map a genotype $g \in G$ to a polyomino $p \in \mathcal{P}_\mathcal{G}$ for a given grammar $\mathcal{G}$ (*i.e.*, a genotype-phenotype mapping function $\phi : G \to \mathcal{P}_\mathcal{G}$ corresponding to the DEVELOP$(\cdot, \mathcal{G}; c, o)$ function of Algorithm 1), we can solve problems of optimization over $\mathcal{P}_\mathcal{G}$ using evolutionary computation (EC), specifically with an EA. Since we have mapping variants for different types of genotype, we can use any EA that best matches the corresponding G. For example, an evolutionary strategy (ES) [31] is suitable for the reals(l) representation, while a genetic algorithm (GA) is appropriate for bits(l), possibly incorporating a linkage-exploitation mechanism [79] as in [50].

To prove the generality of our representation with respect to the EA, in this work we perform the experiments using different EAs, some of them being agnostic with respect to the genotype space G (namely, GA, MAP-elites (ME), and non-dominated sorting genetic algorithm II (NSGA-II)), some being constrained to continuous optimization, *i.e.*, $G = \mathbb{R}^l$, (namely, ES and particle swarm optimization (PSO)). We briefly describe these EAs in the following sections.

3.4.1 Genetic algorithm (GA)

GA is a widely adopted EA with broad applicability and very few requirements—we here consider a rather standard version with a simple modification to enhance diversity.

Given an objective function $f : \mathcal{P}_\mathcal{G} \to \mathbb{R}$ (assuming minimization problems, without loss of generality), GA evolves polyominoes with two variation operators (mutation and crossover based on G). The algorithm employs tournament selection for parent selection and allows overlapping between parents and offspring.

The process begins with the initialization of a population P of n_{pop} individuals, generated using a G-specific procedure. Then, until a termination criterion is met, it iterates through the following steps:

(1) It generates $r_{\text{x-over}} n_{\text{pop}}$ offspring via crossover. Each child is produced by selecting two parents from P with tournament selection (of size n_{tour}) and applying a crossover operator.
(2) It generates $(1 - r_{\text{x-over}}) n_{\text{pop}}$ new individuals via mutation, selecting the parent through tournament selection.
(3) It merges all newly generated individuals with the parents, hence obtaining a population P with $2n_{\text{pop}}$ individuals.
(4) It applies truncation selection to P, retaining the best n_{pop} individuals according to the objective function f.

At the end of the process, GA returns the individual, *i.e.*, the polyomino, with the best objective value.

Whenever a new individual is generated (*i.e.*, in the initial building of the population or through the application of a variation operator), if the genotype is already present in the population, it is discarded and a new one is generated. This way, the genotypic uniqueness is enforced.

Representation-specific components

To initialize the population, namely each genotype in it, we simply generate each g by sampling each one of its elements 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 each i) for structured(l, n_{rec}).

As mutation, we use point-mutation, which 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 standard deviation σ_{mut}: given a parent genotype $\boldsymbol{g} \in \mathbb{R}^l$, we produce the child $\boldsymbol{g}'$ as $\boldsymbol{g}' = \boldsymbol{g} + \boldsymbol{\beta}$, where $\boldsymbol{\beta} \sim N(\boldsymbol{0}, \sigma_{\text{mut}} \boldsymbol{I})$, $\boldsymbol{I}$ being the $l \times l$ identity matrix.

As crossover, we use uniform crossover, which selects each element of the child genotype from one of the parents with equal probability, for ints(l, b), bits(l), and structured(l, n_{rec}). For reals(l), we use the segment geometric crossover followed by the Gaussian mutation: given two parent genotypes $\boldsymbol{g}_1, \boldsymbol{g}_2 \in \mathbb{R}^l$, we produce the child $\boldsymbol{g}'$ as $\boldsymbol{g}' = \boldsymbol{g}_1 + \alpha(\boldsymbol{g}_2 - \boldsymbol{g}_1) + \boldsymbol{\beta}$, where $\alpha \sim U([0, 1])$ and $\boldsymbol{\beta} \sim N(\boldsymbol{0}, \sigma_{\text{mut}} \boldsymbol{I})$.

3.4.2 MAP-elites (ME)

ME [57] is a form of QD optimization [8] which uses an archive for storing the population of individuals, which are indexed by some user-provided *descriptors*. Besides favoring diversity, the archive can also be visualized, revealing insights about the characteristics that make solutions successful.

In this work, we use a standard version of ME and assume that there are two descriptors defined over polyominoes, $d_1 : \mathcal{P}_{\mathcal{G}} \to \mathbb{R}$ and $d_2 : \mathcal{P}_{\mathcal{G}} \to \mathbb{R}$. We use a $n_{\text{bins}} \times n_{\text{bins}}$ grid as substrate for the archive, hence holding at most n_{bins}^2 solutions: given

a polyomino p, we can map it to one cell of the archive by quantizing $(d_1(p), d_2(p))$ using, for each descriptor, a range $[d_{j,\min}, d_{j,\max}]$ and n_{bins} equally sized bins, with $j \in \{1, 2\}$. ME requires also a population initialization procedure and a mutation operator, defined over G: for these components, we make the same choices of GA.

As for GA, the process begins with the initialization of a population of n_{pop} individuals, generated using the G-specific procedure. Each one of the individuals is mapped to a cell of the archive through descriptors: if the cell is empty, it is inserted in the archive, otherwise, it replaces the existing individual only if the latter is worse than the new one (measured using the objective function f).

Then, until some stopping criterion is met, ME iterates the following two steps. First, it randomly selects n_{batch} individuals from the archive (randomly with repetition) and mutate them obtaining an offspring. Second, for each new individual in the offspring, it maps it to a cell (that might be different than the parent one) and attempts to insert it in the archive, as for the initial population.

When the termination criterion is met, the archive contains individuals which are good according to f and diverse according to d_1 and d_2.

3.4.3 Non-dominated sorting genetic algorithm II (NSGA-II)

NSGA-II [12] is a popular EA tailored to multi-objective optimization. In this work, we use it for optimizing the body of VSRs, which is a polyomino, to be fast in locomotion and energy-efficient, hence defining a bi-objective optimization problem (see Section 4.3.2).

Internally, our NSGA-II works as GA and differs only in the way individuals are compared during the selection phases (for reproduction and for survival). Since in a multi-objective problem the objective function $f : \mathcal{P}_{\mathcal{G}} \to \mathbb{R}^k$ is multivariated, there is not a natural total order in the objective space $\mathbb{R}^k$. NSGA-II overcomes this issue by first comparing individuals based on the Pareto-front they lie on (the lower, the better), then by considering the crowding distance, *i.e.*, the Euclidean distance in the objective space $\mathbb{R}^k$ of the individual to other individuals (the greater, the better). This way, NSGA-II can better cover the approximated Pareto-front.

We use NSGA-II with the same initialization procedure and variation operators of GA; we also employ the same simple mechanism for enforcing the genotypic diversity in the population.

3.4.4 Evolutionary strategy (ES)

ES is a rather large family of EAs working on numerical search spaces, *i.e.*, for $G = \mathbb{R}^l$. We here considered a basic variant that works as follows.

After having initialized a population of n_{pop} individuals as in GA, our ES iterates over the following steps until a termination criterion is met. First, it takes the first $\rho_{\text{parent}} n_{\text{pop}}$ individuals based on their ranking dictated by the objective function f. Then, it computes their mean $\boldsymbol{\mu} \in \mathbb{R}^l$. Finally, it forms a new population by generating new $n_{\text{pop}} - 1$ individuals by applying, for obtaining each one, the Gaussian mutation on $\boldsymbol{\mu}$ and retaining the best individual of the previous population.

We use NSGA-II with the same initialization procedure of GA.

3.4.5 Particle swarm optimization (PSO)

Differently from the previously presented methods, PSO does not take inspiration in the natural evolution. Instead, it performs optimization over $G = \mathbb{R}^l$ by seeing candidate solutions as particles that move in the search space driven by a local (called cognitive) and a global (called social) force.

Here we used the standard version of PSO presented in [10], which iteratively refines a swarm of n_{pop} particles, until a termination criterion is met, governing their movement based on the cognitive parameter w, the cognitive acceleration coefficient ϕ_{particle}, and the social acceleration coefficient ϕ_{global}.

For the initialization of the swarm, we use the same procedure of GA.

4 Experiments and results

We performed several 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 that optimize one or more given objectives while adhering to the given grammar. For the latter goal, we also wanted to test different EAs (see Section 4.3.1), to prove the generality of our approach, and different objectives, *i.e.*, ways to measure the quality of the evolving polyominoes. In particular, we considered the challenging case of evolutionary optimization of VSRs (see Section 4.3.2).

Regarding the representations, we experimented with bits(l), ints($l, 4$), ints($l, 16$), reals(l), and structured($l, 2$), using different values for the genotype length l.

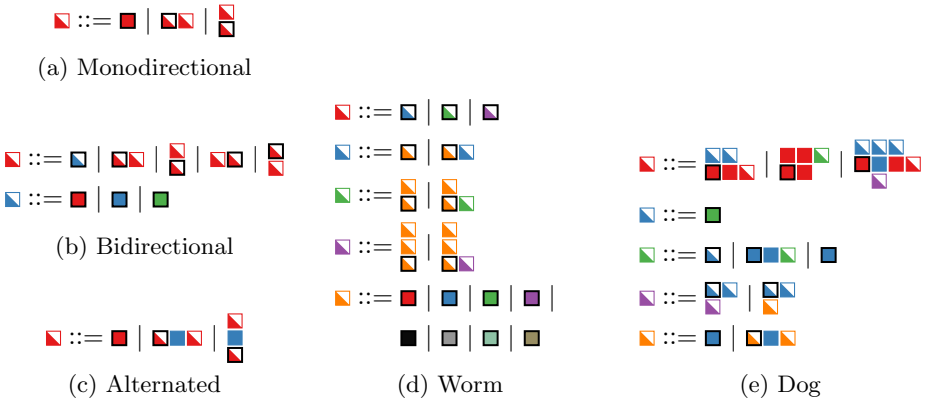
Representation properties

When comparing variants and representation, we focused on analyzing quantitatively some properties of the representation, since they allow to characterize how the search process operates [48, 71]. Specifically, we consider the following quantitative properties, which we measured experimentally:

Validity measures the degree to which a genotype maps to a valid phenotype. Given a set $G' \subseteq G$ of genotypes, we applied our development algorithm to obtain the corresponding bag P of phenotypes and computed the validity as $\frac{1}{|G'|}|\{p \in P : p \neq \varnothing\}|$.

Uniqueness measures the degree to which different genotypes are mapped to distinct phenotypes. Given a set $G' \subseteq G$ 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, as it is a set.

Locality 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



(a) Monodirectional

(b) Bidirectional

(c) Alternated

(d) Worm

(e) Dog

Fig. 6: The PoCFGs considered in the experiments. Half colored squares ◪ represent non-terminal symbols, while fully colored squares ■ represent terminal symbols. On the right-hand-side, the thick black border ■ marks the reference cell within a referenced polyomino. e.g., for the Dog grammar, $N = \{◪, ◪, ◪, ◪, ◪\}$, $T = \{■, ■, ■\}$, $n_1 = ◪$, and there are $|\mathcal{R}| = 11$ production rules.

matrices. We defined d_P as the Hamming distance between a pair of polyominoes after having translated them in order to have coincident centers of mass—when one of the two polyominoes is invalid, *i.e.*, $\varnothing$, we set the distance to the size of the other; if both are $\varnothing$, we set the distance to zero. For d_G, we used the Hamming distance for bits(l), ints(l, b), and structured$(l, 2)$, and the Euclidean distance for reals(l).

For all properties, higher values indicate better characteristics.

We performed our experiments with five PoCFGs, shown in Figure 6. They differ in the number $|\mathcal{R}|$ of rules, the number $|T|$ of terminals, and the number $|N|$ of non-terminals.

4.1 Comparison of development variants

To compare the six variants of our development algorithm (obtained by combining the three sorting criteria and the two overwriting flag values) we used the bits(l) representation, with $l \in \{10, 15, \ldots, 245, 250\}$. We conducted similar experiments for the other representations and observed qualitatively similar findings. For each l value, we generated a set G' of 5000 genotypes (and their corresponding phenotypes) to measure validity and uniqueness, and a sequence G' of 1000 genotypes to measure locality: in both cases, we built each genotype by randomly sampling each element in the proper domain based on the specific representation. Figure 7 presents the results of this experiment.

We first observe that differences in the measured properties are more apparent across PoCFGs (plot columns) than across variants of the algorithm (line colors).

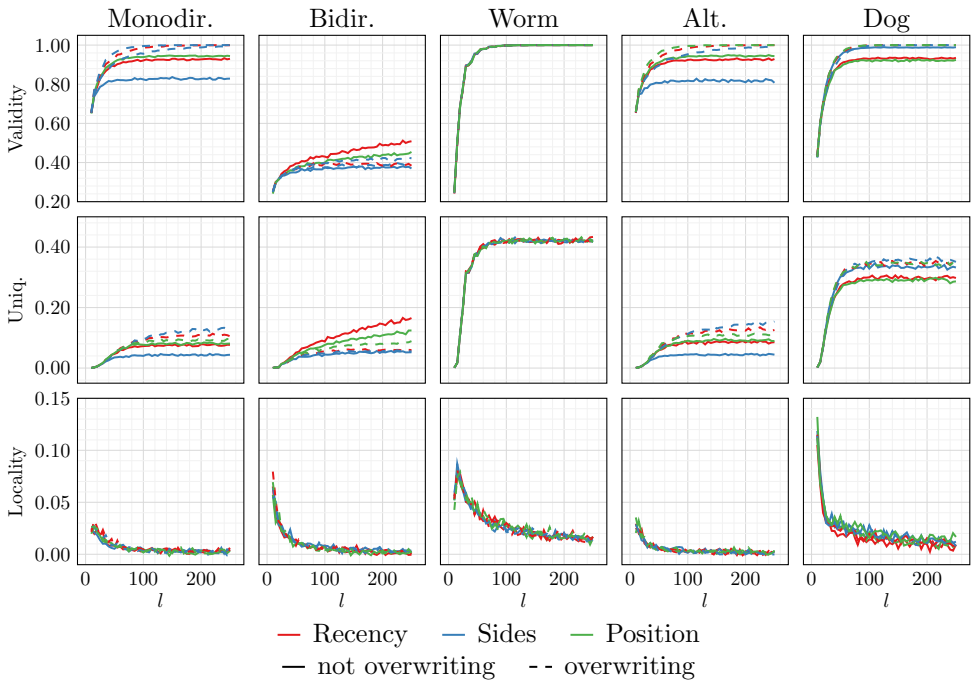


Fig. 7: Representation properties (rows of plots) for the six variants of the development algorithm (line colors and types) measured on the five grammars (columns of plots) with the bits(l) representation.

This suggests that the selection of 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 [32, 43, 61]. At the same time, it indicates that our development algorithm is robust with respect to its parameters.

Looking at the validity plots (first row), it is possible to see that overwriting generally leads to a higher number of valid polyominoes. With all the grammars except for the Bidirectional, the validity reaches its maximum for most of the combinations with overwriting when l is sufficiently large. The lower validity observed with Bidirectional might be related to the fact that there is a higher chance of selecting a non-terminal rather than a terminal, compared to the other grammars.

Concerning uniqueness, Figure 7 suggests that the Sides criterion tends to result in lower uniqueness, whereas Recency in higher uniqueness. No consistent differences are observed between the variants with and without overwriting.

Finally, regarding the locality, the results suggest no differences among the variants. The main role is played by l: as genotype length increases, locality decreases. This finding can be explained by the fact that long genotypes may not be fully used in the mapping process: differences in unused parts of two genotypes are not reflected

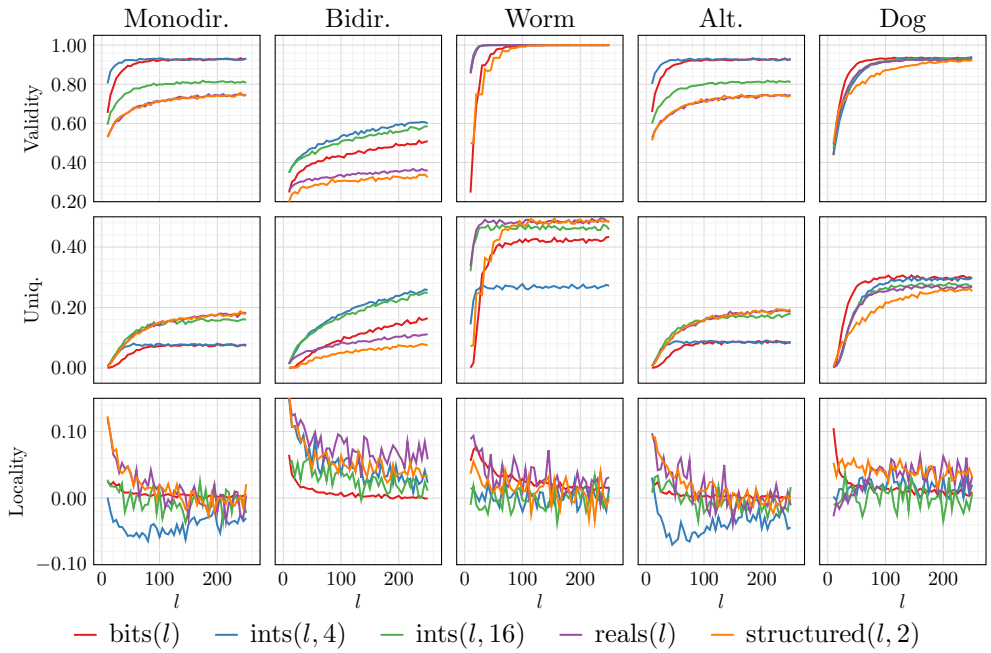


Fig. 8: Representation properties (rows of plots) for the five representations (line color) measured on the five grammars (columns of plots) with the development algorithm based on Recency without overwriting.

in the corresponding phenotypes. This interplay between locality and genotype usage has been previously observed and can be analyzed through visualization tools [54].

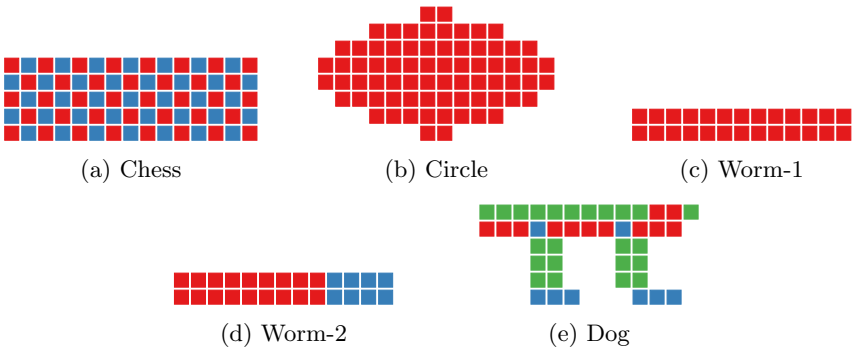
Based on the results of this experiment, we chose the Recency criterion without overwriting for the next experiments. Specifically, we selected the latter parameter value due to its closer alignment with the role of a grammar: describing structural constraints for polyominoes.

4.2 Comparison of representations

We compared the five representations using the same procedure as in the previous experiment, applying the Recency criterion without overwriting. The results are depicted in Figure 8.

As in the previous analysis, the results show that grammar and genotype length l have a greater impact on the representation properties than the representation itself. However, representations exhibit more differences than the development algorithm variants.

The bits(l) and ints$(l, 4)$ representations generally achieve higher validity. structured$(l, 2)$ generates more invalid polyominoes than the other representations, likely due to portions of the genotype being too short—an effect of the n_{rec} parameter. However, larger validity does not always imply a higher number of unique phenotypes:



(a) Chess (b) Circle (c) Worm-1

(d) Worm-2 (e) Dog

Fig. 9: The five target polyominoes.

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

Concerning locality, structured$(l, 2)$ and reals(l) perform, in general, better. While all representations follow a similar trend, bits(l) shows a smoother curve (with, however, low locality).

4.3 Evolutionary optimization of polyominoes

We performed some experiments to verify whether the algorithm proposed can be used inside an EA to solve optimization problems. We considered a set of synthetic optimization problems, where the goal is simply to "approximate" a given target polyomino, and a more realistic problem consisting in optimizing a modular soft robot for performing the task of locomotion (*i.e.*, to run along a flat surface).

4.3.1 Synthetic polyomino optimization problems

We built a set of optimization problems where the goal is to evolve a target polyomino $p^\star$. We used, as objective 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. Intuitively, this objective function measures the *approximation error* of a polyomino p with respect to the target polyomino $p^\star$. We employed this measure of error to facilitate the evolution of the correct shape: namely, we weighted more the shape than the labels of the polyomino, as the former is taken into account in both the component of the objective function, while the labels are used only in the former.

We considered five target polyominoes, shown in Figure 9, and used each of the five PoCFGs of Figure 6 on each target polyomino, hence resulting in 25 optimization problems. We intentionally selected target polyominoes that differ significantly in how well they align with the five PoCFGs. Note that the Dog shape cannot be perfectly reproduced by the Dog grammar due to the misplacement of the rightmost foot in the target polyomino.

Parameter	Value
Number of generations n_{gen} (termination criterion)	200
Population size n_{pop}	100
Tournament size n_{tour}	3
Crossover rate $r_{\text{x-over}}$	0.80
Point-mutation probability p_{mut}	$\frac{1}{l} = 0.002$

Table 1: GA parameters.

We evolved the polyominoes using the GA described in Section 3.4.2 with the parameters shown in Table 1. We employed the bits(500) representation with the Recency criterion and no overwriting: moreover, whenever the polyomino development process concluded with $\varnothing$ (*i.e.*, no mapping), we took a single cell polyomino with the "first" terminal of T as polyomino (*i.e.*, ■ for the Modirectional, Bidirectional, Alternated, and Worm grammars, ■ for the Dog grammar).

We used JGEA [52] for the experiments: we made the code for reproducing the experiments publicly available at https://github.com/ericmedvet/paper-polyomino-grammatical-evolution. For each of the 25 combinations of grammar and target, we performed 50 evolutionary runs, varying the random seed.

Results and discussion

Figure 10 presents the results of these experiments, showing both the approximation error of the best polyomino during the evolution and its size, measured by the 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 targets when using the Worm grammar. Similarly, the Dog grammar greatly outperformed the others on the Dog problem. These findings highlight the importance of well-designed grammars, not only for enforcing structural constraints, but also for embedding domain-specific knowledge to the problem. Figure 11 illustrates the evolution of the best individual across generations when using the Dog or Monodirectional grammar to evolve the Dog shape, for the best respective run. While the shape evolved with the Monodirectional grammar bears some resemblance to the target polyomino, the lack of necessary label colors (terminal symbols) in the grammar makes it impossible to fully reconstruct the Dog shape. Again, these figures confirm the practical importance of being able to conveniently incorporate some domain knowledge in the optimization process through a grammar.

In contrast, when using a grammar not specifically designed to the target, such as the Alternated grammar applied to the Circle, Worm-1, Worm-2, and Dog problems, the EA often became trapped in local minima after only a few iterations. This is evidenced by the unchanged size of the fittest individual, indicating the EA difficulty in effectively exploring the solution space under these conditions.

Overall, these experiments show that it is possible to evolve a polyomino towards a predefined target while ensuring it adheres to specific constraints encoded in a user defined PoCFG.

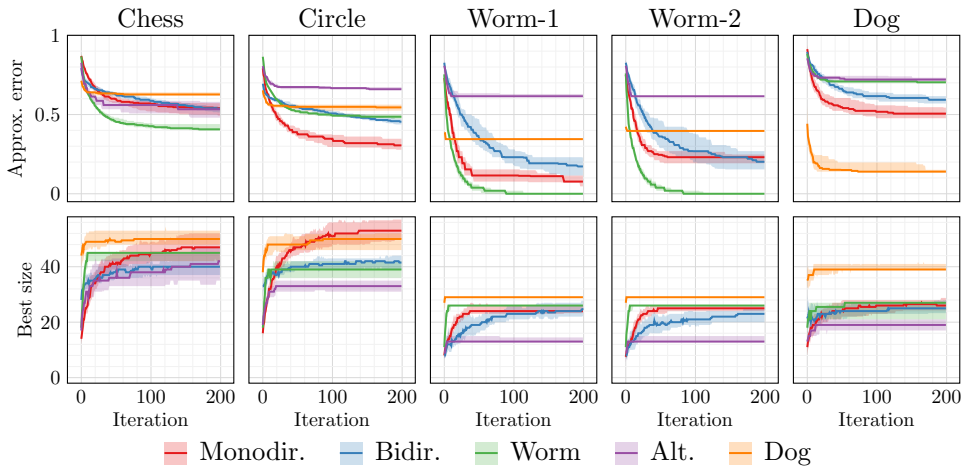


Fig. 10: Approximation error and size (row of plots) of the best polyomino during the evolution for the five problems (column of plots) using the five grammars (color line) with the development algorithm based on Recency without overwriting and the bits(500) representation. The shaded area corresponds to the interquartile range, the line to the median across 50 runs.

More EAs

In order to validate the generality of our approach for optimizing over the space of polyominoes, not only with respect to the representation, but also with respect to the optimization algorithm, we tackled the synthetic polyomino optimization problem with four EAs: GA, ES, PSO, and ME. For these experiments, since ES and PSO require a numerical genotype space, we used the reals(200) representation (with the Recency criterion and without overlapping, as above), where $G = \mathbb{R}^{200}$. Moreover, we considered only the Circle (see Figure 9b) and Dog (see Figure 9e) targets and we used only the Monodirectional, Worm, and Dog grammars (see Figure 6).

We set the parameters for the four EAs as shown in Table 2 and we performed 50 evolutionary runs for each combination of EA, target, and grammar. As the four EAs perform a different number of evaluations of the objective function f at each iteration, we here used the overall number n_{eval} of evaluations, rather than the number n_{gen} of iterations, as termination criterion.

For ME we employed two descriptors aimed at capturing two key features of the shape of a polyomino (*i.e.*, disregarding the labels), elongation and compactness, which have already been used for characterizing the shape of VSR [60, 68] morphologies.

Elongation measures, intuitively, the ratio between the length and width of a polyomino when rotated in the direction of maximum length. For computing it, we first consider the smallest ellipse which encloses the polyomino and then take the ratio of the focal distance on the major axis length. The elongation is defined in $[0, 1]$ and is 0 for non-elongated polyominoes (like, e.g., a square) and ≈ 1 for very long polyominoes (like, e.g., a long row of elements).

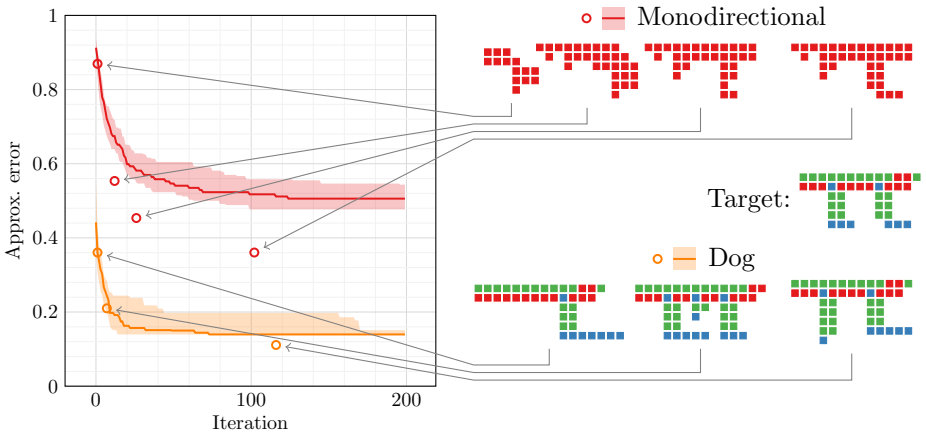


Fig. 11: Representative snapshots (handpicked) at different generations of the best individual evolved to match the polyomino target Dog (right), using the Monodirectional grammar (top) or the Dog grammar (bottom). In the plot the line and the shaded area correspond to the median and interquartile range across 50 runs; the markers are relative to the best run.

Parameter	Value	EAs
Number of evaluations n_{eval} (termination criterion)	20 000	GA, ES, PSO, ME
Population size n_{pop}	100 30	GA,PSO, ME ES
Batch size n_{batch}	100	ME
Number of bins per descriptor n_{bins}	10	ME
Tournament size n_{tour}	3	GA
Crossover rate $r_{\text{x-over}}$	0.80	GA
Gaussian mutation σ_{mut}	0.35	GA, ME, ES
Fraction of parents ρ_{parent}	$\frac{1}{3}$	ES
Cognitive parameter w	0.8	PSO
Cognitive acceleration coefficient ϕ_{particle}	1.5	PSO
Social acceleration coefficient ϕ_{global}	1.5	PSO

Table 2: EAs parameters.

Compactness measures, intuitively, how much a polyomino is "filled". For computing it, we first consider the convex hull enclosing the polyomino and then take the ratio between the number of elements in the polyomino and those in the convex hull. The compactness is defined in $[0, 1]$ and is ≈ 0 for non-compact polyominoes (like, e.g., an "empty" rectangle) and 1 for compact polyominoes (like, e.g., a "full" rectangle). For using it as a descriptor in ME, we only consider the $[0.5, 1]$ sub-range of the natural domain of compactness, as lower values are hard to obtain with small polyominoes.

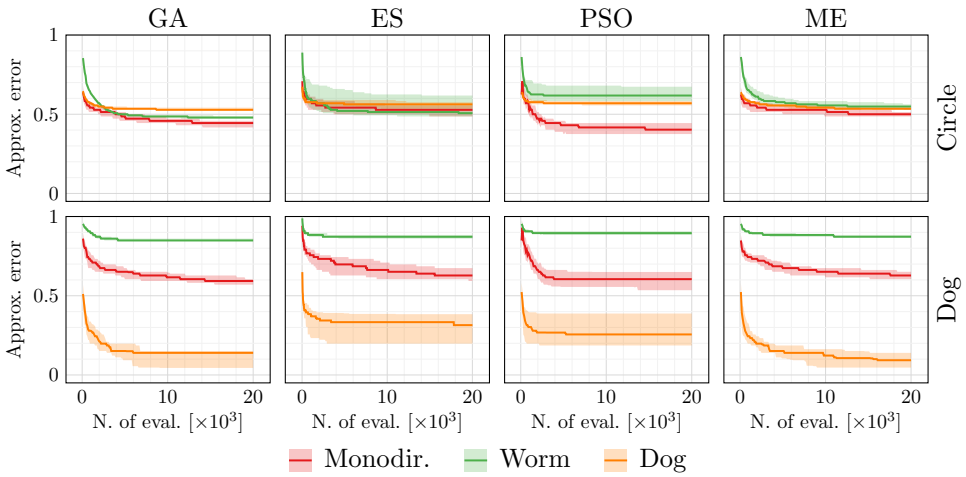


Fig. 12: Approximation error of the best polyomino during the evolution for the two problems (row of plots) using the five grammars (color line) with four EAs (column of plots) employing the development algorithm based on Recency without overwriting and the reals(200) representation. The shaded area corresponds to the interquartile range, the line to the median across 50 runs.

Results. Figure 12 presents the result of this experiment, showing the evolution of the approximation error during the optimization.

The figure confirms that also in this case the most important factor for successfully approximating a target polyomino is the grammar. For the Dog target, polyominoes evolved with the Dog grammar are always much better than those evolved with the other grammars, regardless of the EA employed for the evolution—we remark that the Monodirectional grammar can only produce polyominoes consisting only of ■ elements. Conversely, and consistently with previous results, for the Circle target, none of the EAs is able to produce polyominoes which well approximate the target.

Beyond the confirmation of the importance of the grammar, Figure 12 shows that there are some minor differences among EAs. For the Circle target, PSO achieves a lower approximation error, but only with the Monodirectional grammar. We hypothesize that this is the combined effect of the peculiar fitness landscape induced by the pairing of the grammar and target and the ability of this optimization method to explore and exploit at the same time through the cognitive and social acceleration coefficients, respectively.

For the Dog target, GA and ME score significantly better than ES and PSO (with the Dog grammar). While it is hard to give a sharp interpretation of this finding, we again hypothesize that the fitness landscape plays a key role, namely its ruggedness. Interestingly, with ME we were able to evolve the closest polyominoes to the Dog target, better than those found with GA, also with the bits(500) representation.

In Figure 13 we show the final archives (*i.e.*, those obtained at the end of the optimization) for the first run (the one with seed 1) for the three grammars and two

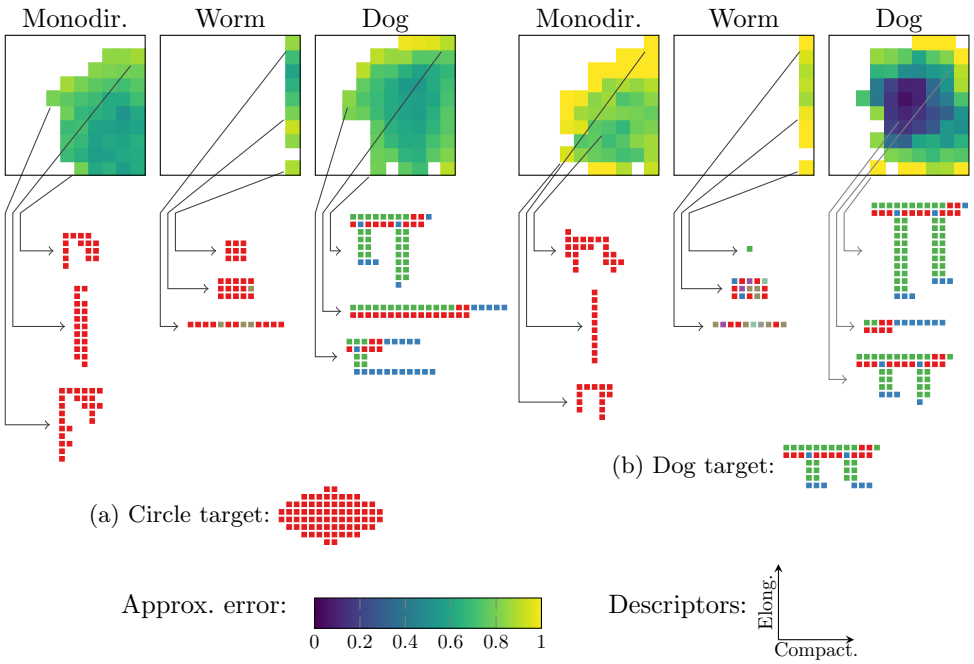


Fig. 13: The ME archive at the end of the run with seed 0 for the Circle (13a) and the Dog (13b) problems using three grammars (row of plots). For each archive, we show three handpicked individuals.

target polyominoes. The figure also shows three polyominoes for each archive, highlighting their position in the archive. It is possible to appreciate that one can restrict the search for polyominoes to those adhering a given grammar and, if the EA permits it, as for the case of ME, favor their diversity over some other user-provided axes (here compactness and elongation). However, the degree to which diversification can be achieved depends on the grammar. With the Worm grammar, all the evolved polyominoes are compact "by design" and the search can only impact on the elongation. With the Dog grammar, all the evolved polyominoes are "Dog-shaped" and the evolution found some with low elongation and compactness (longer legs) or high elongation and compactness (no legs, long trunk). Indeed, with this grammar there is an inherent interplay between elongation and compactness, which is made apparent by ME.

4.3.2 Evolving simulated modular soft robots

In order to further test the ability of our approach to evolving polyominoes adhering a grammar, we considered the more realistic case of the optimization of the morphology of (simulated) modular soft robots—namely VSRs—required to perform some task. As described earlier, the body of a VSR can be described by a labeled polyomino, with labels corresponding to voxel types. Consistently with the rest of this study, we experimented with 2-D VSRs, which have been widely used for research in evolutionary

robotics [6, 9, 49]. Although 3-D counterparts, can be physically fabricated [34, 41], they fall outside the scope of this work, as their bodies are not 2-D polyominoes.

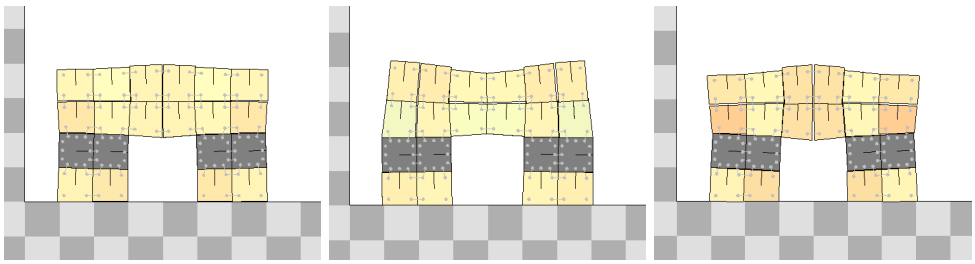
In the next sections, we first provide a brief background on VSRs, then we present our experiments and discuss the results.

Voxel-based soft robots (VSRs)

A VSR is an assembly of soft modules (commonly called voxels, despite being 2-D, in the evolutionary robotics field), each with the ability to actively expand or contract its size. For our simulated 2-D VSRs, each module is a square. Modules are rigidly linked together at the vertices with each adjacent module. We simulate softness, in discrete time, through a compound of masses placed at the vertices of the module and spring-damper systems (SDSs) between masses [49]. We model active expansion or contraction of the module by changing the rest-length of the SDSs: in this work we allow for SDSs on different sides of each module to act independently, hence modeling the capability of the module to actively deform itself. With respect to the reference system of the module, we denote by $\rho_N^{(k)}, \rho_E^{(k)}, \rho_S^{(k)}, \rho_W^{(k)}$ the target relative length of the top side (N), right side (E), bottom side (S), and left side (W) at time step k: for each of them, 1 means that the module side is required to stay at nominal length, a value < 1 means contraction, a value > 1 means expansion. During the simulation, the actual shape of each module depends on its softness, the external forces applied on the module, and the module target relative side lengths. By replacing the SDSs with rigid links, we can model rigid modules, *i.e.*, squares whose area never changes.

The morphology of the VSR, *i.e.*, the way modules are assembled together, can be described by a polyomino. The controller of the VSR is in charge of determining how the area of each module changes over time, *i.e.*, of determining $\rho_N^{(k)}, \rho_E^{(k)}, \rho_S^{(k)}, \rho_W^{(k)}$. While in other works there is a clear distinction between the morphology and the controller of a VSR, often called respectively the body and the brain [19, 20], in this work we assume that the controller is indeed part of each module and hence is distributed across the morphology. Namely, we assume that there are six kinds of module that differ in how they change their shape over time:

- passive hard (PH) modules (from now on denoted with ■), whose area never changes;
- passive soft (PS) modules (■), whose area changes only based on external forces, *i.e.*, $\rho_N^{(k)} = \rho_S^{(k)} = \rho_E^{(k)} = \rho_W^{(k)} = 1$;
- active horizontal (AH) modules (■), whose top and bottom side target lengths change periodically following a sinusoidal function with a frequency of 1 Hz, *i.e.*, $\rho_N^{(k)} = \rho_S^{(k)} = 1 + \rho_{\max} \sin(2\pi k \delta t)$ and $\rho_E^{(k)} = \rho_W^{(k)} = 1$;
- active vertical (AV) modules (■), whose right and left side target lengths change periodically following a sinusoidal function with a frequency of 1 Hz, *i.e.*, $\rho_E^{(k)} = \rho_W^{(k)} = 1 + \rho_{\max} \sin(2\pi k \delta t)$ and $\rho_N^{(k)} = \rho_S^{(k)} = 1$.
- active sinusoidal (AS) modules (■), for which all sides target length changes periodically following a sinusoidal function with a frequency of 1 Hz, *i.e.*, $\rho_N^{(k)} = \rho_E^{(k)} = \rho_S^{(k)} = \rho_W^{(k)} = 1 + \rho_{\max} \sin(2\pi k \delta t)$;



(a) $t = 1.0\,\mathrm{s}$ (b) $t = 1.6\,\mathrm{s}$ (c) $t = 2.2\,\mathrm{s}$

Fig. 14: A VSR composed of 20 modules taken at three time steps separated by 0.6 s. The two top rows of modules are composed of active modules, a group of four AV (▦) modules on the left, four AH (▦) in the middle, four AH (▦) on the right; the legs are PH (▪▪) modules; the feet are PS (▪▪) modules. That results in the polyomino ▦▦▦▦▦▪▪▪▪. In the figure, the color of each module represents its current area ratio (with respect to the original area): the closer to red, the smaller (< 1), the closer to white, the larger (> 1); gray modules are PH. The short light gray lines at the vertices are the rigid links which link together the modules. The gray object with a checkerboard pattern is the ground, where the VSR is situated.

- active cosinusoidal (AC) modules (■), for which all sides target length changes periodically following a cosinusoidal function with a frequency of 1 Hz, *i.e.*, $\rho_N^{(k)} = \rho_E^{(k)} = \rho_S^{(k)} = \rho_W^{(k)} = 1 + \rho_{\max} \cos(2\pi k \delta t)$ (*i.e.*, AC modules contract in counter-phase with respect to AS modules).

Based on the literature [53] and previous experiments, we set the maximum relative side length change to $\rho_{\max} \approx 0.095$ and the time step for the simulation to $\delta t = \frac{1}{60}$ s. Figure 14 shows a VSR composed of 20 modules where four types of modules are represented.

We remark that the kind of control used in these experiments is open-loop, as the voxels determine their sides target relative length without considering any input from the environment, hence not exploiting any perception or proprioception. Data-driven controller synthesis techniques exist in general for both open- and closed-loop controllers [66, 67]. For VSRs, periodic signals are often used as open-loop controllers (as in this case) and artificial neural networks are often used as closed-loop controllers [77]. Recently, some symbolic artifacts, e.g., graphs or ensembles of regression trees have also been explored, for favoring transparency [51, 59]. Interestingly, due to their soft bodies which induce complex dynamics, VSRs can exhibit quite elaborated behaviors even when employing simple open-loop controllers: this is a form of morphological computation [58].

Optimization for efficient locomotion

Task. We considered the task of directed locomotion.

In this task, we placed the VSR on a flat terrain and we performed a simulation lasting 30 s. At the end of the simulation, we measured the average speed v_x of the VSR along the x-axis and the average actuation power p. For the former, we considered the displacement of the center of mass of the VSR between $t = 0$ and $t = 30$ s.

For the average actuation power p, we proceeded as follows. At every time step k and for each side of every voxel, we considered the current relative length $\hat{\rho}^{(k)}$ (computed as the ratio between the current length and the nominal length of the SDS of the voxel side) and the target relative length $\rho^{(k)}$. Then, we accounted for an amount of energy $e^{(k)}$ spent on that voxel side as follows:

$$e^{(k)} = \begin{cases} (\hat{\rho}^{(k)} - 1)(\rho^{(k)} - 1), & \text{if } \hat{\rho}^{(k)} > 1 \wedge \rho^{(k)} > 1 \\ (1 - \hat{\rho}^{(k)})(1 - \rho^{(k)}), & \text{if } \hat{\rho}^{(k)} < 1 \wedge \rho^{(k)} < 1 \\ 0, & \text{otherwise} \end{cases}$$

That is, the energy required to change the length of a side was positive only if it was being further expanded when already expanded or further contracted when already contracted. We define the average power p taken by the VSR as the sum of all the energy amounts spent for all the sides during the entire simulation divided by the duration of the simulation.

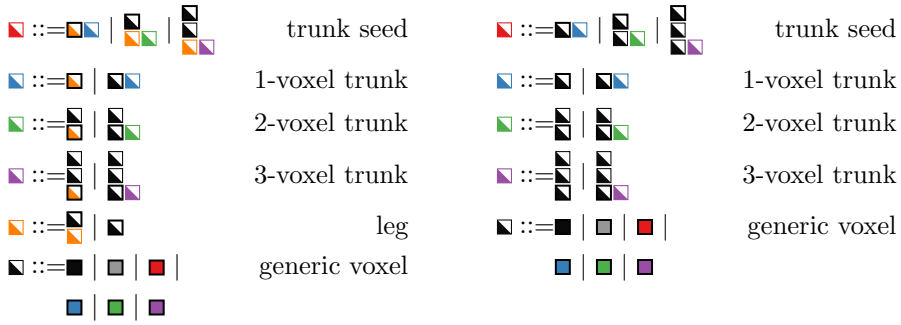
We looked for VSRs which were both fast and energy efficient, *i.e.*, we considered a bi-objective optimization problem consisting in maximizing v_x and minimizing p. Clearly, there is a trade-off between the two objectives: in particular, a VSR composed only of passive voxels (PH or PS) achieves a perfect score in p, but is not actually able to actively move (though it might rotate if its morphology allows it).

Evolutionary algorithm (EA). We used NSGA-II (see Section 3.4.3) for tackling the bi-objective optimization problem of the efficient locomotion of VSRs.

We remark that one could use a different approach for obtaining VSRs which are fast and, and the same time, efficient. For example, based on the observations we did in the previous sections, we could employ ME using a behavioral descriptor (the average power p) and a morphological descriptor (e.g., the number of voxels in the VSR). Indeed, the combined usage of descriptors operating on different aspects of a VSR proved to be an effective way to do QD search for this kind of embodied agents [60].

Representation. We employed a representation of VSRs based on our approach. In particular, we (a) designed two PoCFGs, $\mathcal{G}_{\text{biped}}$ and $\mathcal{G}_{\text{worm}}$, describing two sets of VSRs to be searched for; and (b) we chose the bits(1024) representation (see Section 3.3.3), *i.e.*, we used the genotype space $G = \{0, 1\}^{1024}$, with the Recency criterion and no overwriting (see Section 3.3.2).

We show the two PoCFGs $\mathcal{G}_{\text{biped}}$ and $\mathcal{G}_{\text{worm}}$ in Figure 15. Both have six terminal symbols, *i.e.*, $T = \{$■, ■, ■, ■, ■, ■$\}$, corresponding to PH, PS, AH, AV, AS, AC voxels, respectively. $\mathcal{G}_{\text{biped}}$ has six non-terminal symbols ($N = \{$◪, ◪, ◪, ◪, ◪, ◪$\}$) and 17 production rules; $\mathcal{G}_{\text{worm}}$ has one fewer non-terminal symbol (◪, corresponding to the seed for the leg) and 15 production rules. Intuitively, $\mathcal{G}_{\text{biped}}$ describes the subset of VSRs whose morphology resembles a biped, *i.e.*, a trunk with two legs. In particular, the horizontal trunk can be one, two, or three voxels wide and with unbounded length; each



(a) $\mathcal{G}_{\text{biped}}$, for biped-like VSRs.

(b) $\mathcal{G}_{\text{worm}}$, for worm-like VSRs.

Fig. 15: The two PoCFGs used in our experiments. The thick black border denotes the reference cell.

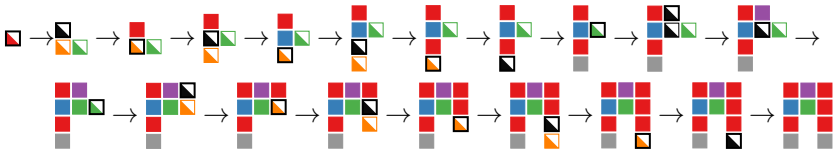


Fig. 16: An example of the development of a VSR adhering to $\mathcal{G}_{\text{biped}}$ (with the Recency sorting criterion and no overwriting). The thick black border denotes the cell being replaced.

of the vertical legs is one voxel wide and with a length ranging from zero to infinite—note that the two legs can be of different length. Similarly, $\mathcal{G}_{\text{biped}}$ describes the subset of VSRs whose morphology resembles a worm, *i.e.*, a rectangle whose length is at least of two voxels and whose width is of either one, two, or three voxels. For both grammars, voxels of any type can be used in every part of the morphology. Hence, $\mathcal{G}_{\text{biped}}$ and $\mathcal{G}_{\text{worm}}$ only constrain the shape of the VSRs the EA will search on. One might impose finer grained constraints by augmenting the PoCFG for specifying that, e.g., biped feet have to be PH, maybe for making the VSR more robust to wear.

In Figure 16 we show an example of the development (with the Recency criterion and no overwriting) of a polyomino adhering to $\mathcal{G}_{\text{biped}}$.

As a comparison baseline, we also employed another representation for VSRs which does not constrain the robots to have a specific, grammar-based morphology. In this representation, which we called *grid-based*, the genotype space is $G = \{0, 1, 2, 3, 4, 5, 6\}^{24}$ and the mapping process works as follows. Given an int string $g \in G$, we first reshape it to a 6×4 matrix, then we consider the largest polyomino

Parameter	Value
Number of generations n_{gen} (termination criterion)	200
Population size n_{pop}	100
Tournament size n_{tour}	3
Crossover rate $r_{\text{x-over}}$	0.80
Point-mutation probability p_{mut}	$\frac{1}{l} \approx \begin{cases} 0.001 & \text{for grammar-based} \\ 0.042 & \text{for grid-based} \end{cases}$

Table 3: NSGA-II parameters.

in the matrix formed by non-zero elements (*i.e.*, the largest single connected component), and finally we map each matrix element greater than zero to a corresponding value in {■, ■, ■, ■, ■, ■}.

Summarizing, we compared three approaches for tackling the efficient locomotion optimization problem. All of them use NSGA-II as EA and search in the space of (a subset of) VSRs. The grammar-based approaches use $\{0, 1\}^{1024}$ as genotype space G; the grid-based approach uses $\{0, 1, 2, 3, 4, 5, 6\}^{24}$ as genotype space G. For all, we used the uniform crossover and the point-mutation as genetic operators and the random initialization in the domain of each genotype element for population initialization.

We remark that we are not aware of any previous result (either theoretical or experimental) suggesting which is the best morphology for a VSR for the task of locomotion. A recent research showed that there is an interplay between the "average" morphology obtained through optimization and the nature of the task [68]: e.g., for the locomotion along a descent, circular shapes are favored, as they enable a rolling gait. In particular, there is no evidence that biped-like or worm-like VSRs are more effective than others, regardless of the kind of controller. Nevertheless, these two shapes have been widely used in previous works, mainly for their biological resemblance.

Results and discussion

We executed 30 evolutionary runs for each of the three approaches. We used JGEA for the optimization and 2D-VSR-SIM [49] for simulating the VSRs: we made the code for reproducing the experiments publicly available at https://github.com/ericmedvet/paper-polyomino-grammatical-evolution. We set the parameters of NSGA-II as shown in Table 3.

Figure 17 shows the progression of the speed v_x and power p of the fastest VSR in the population during the evolution for the three representations. We remark that we considered a bi-objective optimization problem: however, we arbitrarily chose to prioritize v_x in this visualization by choosing the fastest VSR. For v_x, the greater, the better; for p, the opposite.

It can be noticed from Figure 17 that VSRs evolved with the grammar-based representations were in general more efficient than those evolved with the grid-based representation. Moreover, bipeds (*i.e.*, $\mathcal{G}_{\text{bided}}$-based VSRs) and "free-form" VSRs (*i.e.*, grid-based ones) were equally fast, whereas worms (*i.e.*, $\mathcal{G}_{\text{worm}}$-based VSRs) were in general much slower. We explain the latter finding by the inherent simpler dynamics of the worm shape. Worms likely require a crawling-like gait for achieving locomotion

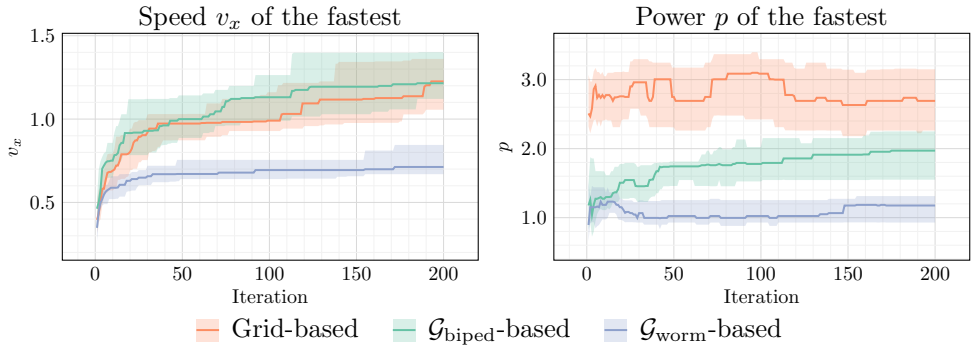


Fig. 17: Average speed v_x (left) and power p (right) of the fastest robot in the population during the evolution, for the three approaches. The shaded area corresponds to the interquartile range, the line to the median across 30 runs.

effectively: however, due to the limitation of the open-loop controller employed in this work, this kind of behavior is hard to achieve.

Concerning the difference in efficiency between bipeds and free-form VSRs, we observe that the two representations corresponded to different search space and different biases in the search. With the $\mathcal{G}_{\text{bided}}$-based representation, we were able to inject in the approach to the optimization problem our knowledge (or, from another point of view, our expectation based on biological resemblance) about the potential suitability of the biped shape. On the other hand, the grid-based representation is more expressive (tough it can evolve VSRs of at most 6×4 voxels) and does not incorporate any bias. With this representation, NSGA-II took longer to reach equally fast VSRs and would have likely taken even longer to improve their efficiency. Note that, with the grid-based representation, the EA might evolve biped-shaped VSRs and also, maybe, shapes allowing a better control. The properties that make a VSR body more or less controllable for a given task have yet to be completely characterized, but in general appear hard to capture [78].

What really matters for our study is that with both grammar-based representations we were actually able to constrain the search to a precisely defined subset of VSRs, which we have conveniently defined through a PoCFG. Indeed, we show in Figure 18 the fastest VSR obtained in each of the 30 repetitions for the three approaches. It can be easily noted that all the VSRs evolved with the grammar-based approaches are actually bipeds—some of them with zero-long legs, some with short trunks—and worms, respectively in the $\mathcal{G}_{\text{bided}}$-based and $\mathcal{G}_{\text{worm}}$-based case. Conversely, robots evolved with the grid-based approach are actually free-form (though always of size of at most 6×4). In order to apply a specific shape constraint, e.g., being biped-like, with the grid-based representation, one would have had to (a) design and implement a non-trivial algorithm for determining whether (or the degree to which) a polyomino is a biped and (b) integrate it in the EA as a hard constraint, a penalization in the one of the two objectives, or maybe a third objective. Arguably, the effort would have been greater than the one required to define $\mathcal{G}_{\text{biped}}$.

(a) Grid-based.

(b) $\mathcal{G}_{\text{biped}}$-based.

(c) $\mathcal{G}_{\text{worm}}$-based.

Fig. 18: The fastest VSR obtained in each one of the 30 evolutionary runs for the three approaches. The videos of these VSRs performing locomotion can be found at https://github.com/ericmedvet/paper-polyomino-grammatical-evolution.

Finally, to further remark the effects of constraining the search using a PoCFG, we show in Figure 19 the values of the objectives for each VSR in the first Pareto-front of the final population obtained with the three approaches. For a few VSRs for each approach, we also show the corresponding morphology.

It can be seen that the three representations were apparently differently able to cover the objective space, *i.e.*, they achieved different v_x-p trade-offs. While in all the three cases the vast majority of evolved VSRs are more efficient than fast, the $\mathcal{G}_{\text{biped}}$- and $\mathcal{G}_{\text{worm}}$-based robots tend to dominate the grid-based ones for low values of v_x. Interestingly, many $\mathcal{G}_{\text{biped}}$-based VSRs in this region of the space are bipeds without legs, *i.e.*, worms—indeed, $\mathcal{P}_{\mathcal{G}_{\text{biped}}} \supset \mathcal{P}_{\mathcal{G}_{\text{worm}}}$. For fast robots the $\mathcal{G}_{\text{worm}}$-based representation is not very effective: the fastest worm achieves an average speed of $v_x \approx 1.13$, whereas the fastest biped and free-form VSR score, respectively, 1.58 and 1.53. However, for the same speed, the free-form VSR is much larger (and hence consumes more energy for the gait) than the biped.

5 Conclusions

In this work, we introduced the concept of polyomino context-free grammars (PoCFGs) and a novel algorithm to develop polyominoes that meet predetermined requirements, defined by a PoCFG. The development algorithm can be driven by a source of information (a list of integers, bits, or similar data structures). This way it can be integrated within an evolutionary algorithm (EA) where the genotypes of individuals are the

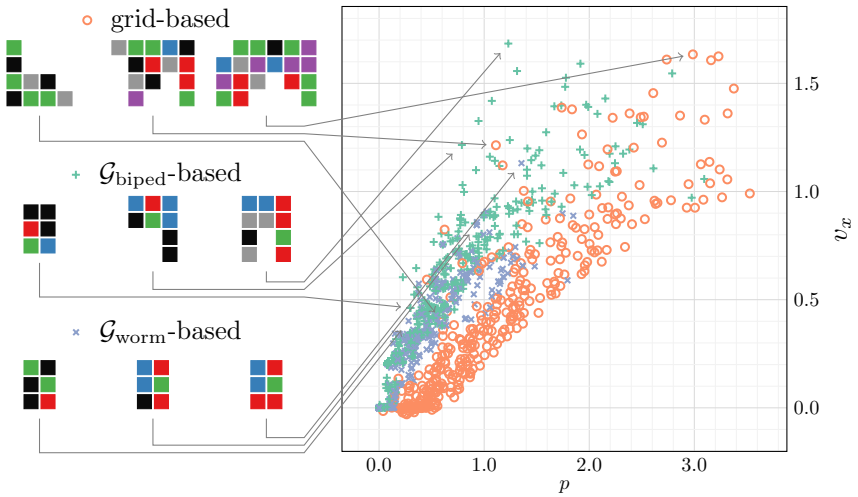


Fig. 19: Average speed v_x and power p of each VSR in the first front of the final population of all the 30 runs for the three approaches. For three VSRs for each approach, we show the morphology.

source of information used by the development algorithm. In brief, the latter can be integrated in any EA to solve optimization problems where the search space is a set of labeled polyominoes defined by a grammar.

We performed three experimental campaigns and discussed the results. In the first one, we characterized experimentally the impact of the key components of our development algorithm. Namely, we considered four different kinds of genotypes and two ways of developing the polyomino and compared them in terms of well-established property for representations: validity, uniqueness, and locality. In the second set of experiments, we defined a set of synthetic optimization problems where the goal was to evolve a polyomino with a given shape: this way, we showed that our grammar-based representation can actually be used by "any" EA in the context of evolutionary optimization—we experimented with genetic algorithm (GA), evolutionary strategy (ES), particle swarm optimization (PSO), and MAP-elites (ME), hence validating the generality of our approach with respect to the optimization algorithm being employed. Finally, in the third experiment, we considered a more realistic optimization problem where the goal was to find a modular soft robot (namely, a voxel-based soft robot (VSR), whose body can be described by a labeled polyomino) that runs fast and efficiently.

Our results showed that it is actually possible to use general-purpose EAs to solve optimization problems with polyominoes. Nevertheless, we also found that our representation, like other grammar-based representations, tend to suffer from diversity and locality issues. This finding aligns with existing literature on grammar-guided genetic programming (G3P). However, we also confirmed how convenient is the possibility of sharply defining constraints on the search space by means of a grammar. Such a

practical advantage might be relevant in other scenarios where the "physical structure" of the solution is important, as in the generation of maps for games [36] or DNA shapes [75].

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 is funded by national funds through FCT – Foundation for Science and Technology, I.P., within the scope of the research unit UID/00326 - Centre for Informatics and Systems of the University of Coimbra, 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.

References

[1] Aigrain, P., Beauquier, D.: Polyomino tilings, cellular automata and codicity. Theoretical Computer Science **147**(1-2), 165–180 (1995)

[2] Ashlock, D.: Cellular Encoding, pp. 381–423. Springer New York, New York, NY (2006), ISBN 978-0-387-31909-4, doi:10.1007/0-387-31909-3_14, URL https://doi.org/10.1007/0-387-31909-3_14

[3] Barequet, G., Ben-Shachar, G.: Counting Polyominoes, Revisited, pp. 133–143 (may 2024), doi:10.1137/1.9781611977929.10, URL https://epubs.siam.org/doi/abs/10.1137/1.9781611977929.10

[4] Barequet, G., Golomb, S.W., Klarner, D.A.: Polyominoes. In: Handbook of Discrete and Computational Geometry, pp. 359–380, Chapman and Hall/CRC (2017)

[5] Bartoli, A., Castelli, M., Medvet, E.: Weighted hierarchical grammatical evolution. IEEE transactions on cybernetics **50**(2), 476–488 (2018)

[6] Bhatia, J., Jackson, H., Tian, Y., Xu, J., Matusik, W.: Evolution gym: A large-scale benchmark for evolving soft robots. Advances in Neural Information Processing Systems **34**, 2201–2214 (2021)

[7] Browne, C.A., Amchin, D.B., Schneider, J., Datta, S.S.: Infection percolation: A dynamic network model of disease spreading. Frontiers in Physics **Volume 9 - 2021** (2021), ISSN 2296-424X, doi:10.3389/fphy.2021.645954, URL https://www.frontiersin.org/journals/physics/articles/10.3389/fphy.2021.645954

[8] Chatzilygeroudis, K., Cully, A., Vassiliades, V., Mouret, J.B.: Quality-diversity optimization: a novel branch of stochastic optimization. In: Black box optimization, machine learning, and no-free lunch theorems, pp. 109–135, Springer (2021)

[9] Cheney, N., MacCurdy, R., Clune, J., Lipson, H.: Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding.

SIGEVOlution **7**(1), 11–23 (Aug 2014), doi:10.1145/2661735.2661737, URL https://doi.org/10.1145/2661735.2661737
[10] Clerc, M.: Beyond standard particle swarm optimisation. In: Innovations and Developments of Swarm Intelligence Applications, pp. 1–19, IGI Global (2012)
[11] Conway, A.: Enumerating 2d percolation series by the finite-lattice method: Theory. Journal of Physics A: Mathematical and General **28**(2), 335 (1995)
[12] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE transactions on evolutionary computation **6**(2), 182–197 (2002)
[13] Delest, M.: Polyominoes and animals: some recent results. Journal of mathematical chemistry **8**(1), 3–18 (1991)
[14] Delest, M.P., Fedou, J.M.: Counting polyominoes using attribute grammars. In: Deransart, P., Jourdan, M. (eds.) Attribute Grammars and their Applications, pp. 46–60, Springer Berlin Heidelberg, Berlin, Heidelberg (1990), ISBN 978-3-540-46666-6
[15] van Diepen, M., Shea, K.: A spatial grammar method for the computational design synthesis of virtual soft locomotion robots. Journal of Mechanical Design **141**(10) (May 2019), ISSN 1528-9001, doi:10.1115/1.4043314, URL http://dx.doi.org/10.1115/1.4043314
[16] Duchi, E., Rinaldi, S.: An object grammar for column-convex polyominoes. Annals of Combinatorics **8**, 27–36 (2004), URL https://api.semanticscholar.org/CorpusID:120909472
[17] Duchon, P.: Q-grammars and wall polyominoes. Annals of Combinatorics **3**(2), 311–321 (1999), doi:10.1007/BF01608790, URL https://doi.org/10.1007/BF01608790
[18] Fekete, S.P., Hendricks, J., Patitz, M.J., Rogers, T.A., Schweller, R.T.: Universal computation with arbitrary polyomino tiles in non-cooperative self-assembly. In: Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, p. 148–167, SODA '15, Society for Industrial and Applied Mathematics, USA (2015)
[19] Ferigo, A., Iacca, G., Medvet, E., Nadizar, G.: Totipotent neural controllers for modular soft robots: Achieving specialization in body–brain co-evolution through hebbian learning. Neurocomputing **614**, 128811 (2025)
[20] Ferigo, A., Medvet, E., Iacca, G.: Optimizing the sensory apparatus of voxel-based soft robots through evolution and babbling. SN Computer Science **3**(2) (Dec 2021)
[21] Fernau, H., Schmid, M.L., Subramanian, K.G.: Two-dimensional pattern languages. In: Workshop on Non-Classical Models for Automata and Applications (2017)
[22] Fortier, J., Goupil, A., Lortie, J., Tremblay, J.: Exhaustive generation of gominoes. Theoretical Computer Science **502**, 76–87 (2013), ISSN 0304-3975, doi:https://doi.org/10.1016/j.tcs.2012.02.032, URL https://www.sciencedirect.com/science/article/pii/S0304397512001843, generation of Combinatorial Structures
[23] Frosini, A., Rinaldi, S., et al.: An object grammar for the class of l-convex polyominoes. Pure Mathematics and Applications (PU.M.A.) **17**(1-2), 97–110

(2006)
[24] 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
[25] Gheorghe, M., Păun, G.: Chapter 3 computing by self-assembly: Dna molecules, polyominoes, cells. In: Krasnogor, N., Gustafson, S., Pelta, D.A., Verdegay, J.L. (eds.) Systems Self-Assembly, Studies in Multidisciplinarity, vol. 5, pp. 49–78, Elsevier (2008), doi:https://doi.org/10.1016/S1571-0831(07)00003-2, URL https://www.sciencedirect.com/science/article/pii/S1571083107000032
[26] Giammarresi, D., Restivo, A.: Two-Dimensional Languages, pp. 215–267. Springer Berlin Heidelberg, Berlin, Heidelberg (1997), ISBN 978-3-642-59126-6
[27] 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)
[28] Grandjean, A., Poupet, V.: L-convex polyominoes are recognizable in real time by 2d cellular automata. In: Cellular Automata and Discrete Complex Systems: 21st IFIP WG 1.5 International Workshop, AUTOMATA 2015, Turku, Finland, June 8-10, 2015. Proceedings 21, pp. 127–140, Springer (2015)
[29] Grimmett, G.: What is Percolation?, pp. 1–31. Springer Berlin Heidelberg, Berlin, Heidelberg (1999), ISBN 978-3-662-03981-6
[30] Guo, M., Shou, W., Makatura, L., Erps, T., Foshey, M., Matusik, W.: Polygrammar: Grammar for digital polymer representation and generation. Advanced Science **9**(23), 2101864 (2022), doi:https://doi.org/10.1002/advs.202101864, URL https://advanced.onlinelibrary.wiley.com/doi/abs/10.1002/advs.202101864
[31] Hansen, N., Arnold, D.V., Auger, A.: Evolution strategies. Springer handbook of computational intelligence pp. 871–898 (2015)
[32] Harper, R.: GE, explosive grammars and the lasting legacy of bad initialisation. In: IEEE Congress on Evolutionary Computation, IEEE (Jul 2010)
[33] He, G., Yang, Q., Fu, F., Kwak, K.S.: Percolation theory aided data diffusion for mobile wireless networks. In: 2012 International Conference on ICT Convergence (ICTC), pp. 61–65 (2012), doi:10.1109/ICTC.2012.6386780
[34] Hiller, J., Lipson, H.: Automatic design and manufacture of soft robots. IEEE Transactions on Robotics **28**(2), 457–466 (2011)
[35] Hunt, A.G., Sahimi, M.: Flow, transport, and reaction in porous media: Percolation scaling, critical-path analysis, and effective medium approximation. Reviews of Geophysics **55**(4), 993–1078 (Nov 2017), ISSN 1944-9208, doi:10.1002/2017rg000558, URL http://dx.doi.org/10.1002/2017RG000558
[36] 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)
[37] Knight, T., Stiny, G.: Making grammars: from computing with shapes to computing with things. Design Studies **41**, 8–28 (2015)
[38] Knuth, D.E.: Dancing links. arXiv preprint cs/0011047 (2000)
[39] 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

[40] 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)

[41] Legrand, J., Terryn, S., Roels, E., Vanderborght, B.: Reconfigurable, multi-material, voxel-based soft robots. IEEE Robotics and Automation Letters **8**(3), 1255–1262 (2023)

[42] Lourenço, N., Pereira, F.B., Costa, E.: Unveiling the properties of structured grammatical evolution. Genetic Programming and Evolvable Machines **17**, 251–289 (2016)

[43] 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)

[44] Manzoor, S., Sheckman, S., Lonsford, J., Kim, H., Kim, M.J., Becker, A.T.: Parallel self-assembly of polyominoes under uniform control inputs. IEEE Robotics and Automation Letters **2**(4), 2040–2047 (2017), doi:10.1109/LRA.2017.2715402

[45] Martin, G.E.: Polyominoes: A guide to puzzles and problems in tiling (1996), URL https://api.semanticscholar.org/CorpusID:123442821

[46] Mason, J.: oeis.org. https://oeis.org/A000105/a000105_1.pdf (2023), [Accessed 14-03-2025]

[47] 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

[48] Medvet, E.: A comparative analysis of dynamic locality and redundancy in grammatical evolution. In: European Conference on Genetic Programming, pp. 326–342, Springer (2017)

[49] Medvet, E., Bartoli, A., De Lorenzo, A., Seriani, S.: 2d-vsr-sim: A simulation tool for the optimization of 2-d voxel-based soft robots. SoftwareX **12**, 100573 (2020)

[50] 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)

[51] Medvet, E., Nadizar, G.: GP for Continuous Control: Teacher or Learner? The Case of Simulated Modular Soft Robots. In: Genetic Programming Theory and Practice XX, pp. 203–224, Springer (2024), doi:10.1007/978-981-99-8413-8_11

[52] 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)

[53] Medvet, E., Nadizar, G., Pigozzi, F.: On the impact of body material properties on neuroevolution for embodied agents: The case of voxel-based soft robots. In: Proceedings of the genetic and evolutionary computation conference companion, pp. 2122–2130 (2022)

[54] 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)
[55] Mégane, J., Medvet, E., Lourenço, N., Machado, P.: Grammar-based evolution of polyominoes. In: Giacobini, M., Xue, B., Manzoni, L. (eds.) Genetic Programming, pp. 56–72, Springer Nature Switzerland, Cham (2024), ISBN 978-3-031-56957-9
[56] Mordvintsev, A., Randazzo, E., Niklasson, E., Levin, M.: Growing neural cellular automata. Distill **5**(2), e23 (2020)
[57] Mouret, J.B., Clune, J.: Illuminating search spaces by mapping elites. arXiv preprint arXiv:1504.04909 (2015)
[58] Müller, V.C., Hoffmann, M.: What is morphological computation? On how the body contributes to cognition and control. Artificial life **23**(1), 1–24 (2017)
[59] Nadizar, G., Medvet, E., Wilson, D.G.: Naturally interpretable control policies via graph-based genetic programming. In: European Conference on Genetic Programming (Part of EvoStar), pp. 73–89, Springer (2024)
[60] Nadizar, G., Medvet, E., Wilson, D.G.: Enhancing adaptability in embodied agents: A multi-quality-diversity approach. IEEE Transactions on Evolutionary Computation (2025)
[61] Nicolau, M., Agapitos, A.: Understanding grammatical evolution: Grammar design. In: Handbook of Grammatical Evolution, pp. 23–53, Springer International Publishing (2018)
[62] 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)
[63] 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
[64] Ota, P.A.: Mosaic grammars. Pattern recognition **7**(1-2), 61–65 (1975)
[65] Packard, N.H., Wolfram, S.: Two-dimensional cellular automata. Journal of Statistical Physics **38**(5), 901–946 (1985)
[66] Pellegrino, F.A., Blanchini, F., Fenu, G., Salvato, E.: Closed-loop control from data-driven open-loop optimal control trajectories. In: 2022 European Control Conference (ECC), pp. 1379–1384, IEEE (2022)
[67] Pellegrino, F.A., Blanchini, F., Fenu, G., Salvato, E.: Data-driven dynamic relatively optimal control. European Journal of Control **74**, 100839 (2023)
[68] 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)
[69] Pinto, D.E.P., Araújo, N.A.M., Šulc, P., Russo, J.: Inverse design of self-folding 3d shells. Phys. Rev. Lett. **132**, 118201 (Mar 2024), doi:10.1103/PhysRevLett.132.118201, URL https://link.aps.org/doi/10.1103/PhysRevLett.132.118201

[70] Prusa, D., Hlavá, V.: 2d context-free grammars: Mathematical formulae recognition. In: Prague Stringology Conference (2006)
[71] Rothlauf, F., Goldberg, D.E.: Redundant representations in evolutionary computation. Evolutionary Computation **11**(4), 381–415 (2003)
[72] Rusin, F., Medvet, E.: How perception, actuation, and communication impact the emergence of collective intelligence in simulated modular robots. Artificial Life **30**(4), 448–465 (2024)
[73] Ryan, C., Collins, J.J., O'Neill, M.: 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)
[74] Sakai, I.: Syntax in universal translation. In: Proceedings of the International Conference on Machine Translation and Applied Language Analysis (1961)
[75] 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)
[76] Subramanian, K., Ali, R.M., Geethalakshmi, M., Nagar, A.K.: Pure 2d picture grammars and languages. Discrete Applied Mathematics **157**(16), 3401–3411 (2009)
[77] Talamini, J., Medvet, E., Bartoli, A., Lorenzo, A.D.: Evolutionary synthesis of sensing controllers for voxel-based soft robots. In: The 2019 Conference on Artificial Life, MIT Press (2019)
[78] Talamini, J., Medvet, E., Nichele, S.: Criticality-driven evolution of adaptable morphologies of voxel-based soft-robots. Frontiers in Robotics and AI **8** (Jun 2021)
[79] 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)
[80] Vanderzande, C.: Lattice Models of Polymers. Cambridge Lecture Notes in Physics, Cambridge University Press (1998)
[81] Vujic, D.: Branched polymers on the two-dimensional square lattice with attractive surfaces. Journal of Statistical Physics **95**(3), 767–774 (1999), doi:10.1023/A:1004507812769, URL https://doi.org/10.1023/A:1004507812769
[82] Whittington, S.G., Soteros, C.E.: Lattice animals: Rigorous results and wild guesses (1990)
[83] Winslow, A.: Staged self-assembly and polyomino context-free grammars. Natural Computing **14**(2), 293–302 (2015)