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

Jessica Mégane

jessicac@dei.uc.pt

University of Coimbra, CISUC/LASI – Centre for Informatics and Systems of the University of Coimbra

Eric Medvet

University of Trieste

Nuno Lourenço

University of Coimbra, CISUC/LASI – Centre for Informatics and Systems of the University of Coimbra

Penousal Machado

University of Coimbra, CISUC/LASI – Centre for Informatics and Systems of the University of Coimbra

Research Article

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

Posted Date: October 9th, 2025

DOI: https://doi.org/10.21203/rs.3.rs-6982435/v1

Additional Declarations: No competing interests reported.

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 [17, 22, 35]. These languages generate 2-D objects, ranging from simple rectangles to more complex shapes like decagons. Among these, polyominoes have attracted significant interest due to their unique properties [4, 20, 34] and diverse applications [25, 36, 56, 57, 62, 64, 70].

Polyominoes [23] are geometric shapes formed by connected unit squares, forming a finite set of cells within a 2-D grid. Also known as lattice animals in the physical [25] and chemical [57] fields, they are widely used to model branched polymers, molecular structures, and percolation processes, offering insights into complex systems [8, 77]. Their mathematical properties have led to extensive study in combinatorial optimization and mathematics [4, 20, 34], but also influenced theoretical formal language research [22, 23]. In many of these fields, it is often crucial to find one or more polyominoes that maximize specific objectives while satisfying predefined structural requirements [2]. A powerful mechanism for describing and generating 2-D shapes is grammars [36, 56, 58, 64, 78].

Grammars formally describe a language through a set a production rules. To generate 2-D shapes, they can be designed either as one-dimensional encoding of two-dimensional structures [78] or as fully two-dimensional representations [58, 64]. While grammars have been used to generate and study the properties of polyominoes [78], to the best of our knowledge, no existing approach employs them to evolve these shapes for optimization purposes. However, grammars have been widely applied in the generation of 2-D pictures [33, 35, 43, 71], finding practical applications in popular tasks such as mathematical formula recognition [36, 56, 64].

Polyominoes can be further enhanced by assigning a label to each individual cell, hence providing additional information within the structure. In a previous work [51], we presented a novel approach for generating labeled polyominoes that adhere to structural requirements defined in a formal way. Specifically, we: (a) defined the concept of polyomino context-free grammars (PoCFGs) as an extension of context-free grammars (CFGs) and (b) proposed a development algorithm that can be used for generating a polyomino adhering to a PoCFGs. Our algorithm constructs these polyomino structures with precise control of the shape of the polyomino and labeling of its cells. When used inside an evolutionary algorithm (EA), it allows solving optimization problems over the space of labeled polyominoes adhering to a given PoCFG. This process only requires the provision of a grammar and a fitness function to the selected EA; notably, it does not require users to provide variation operators that guarantee that varied polyominoes will still adhere to the PoCFG—*i.e.*, operators with the closure property. This fact greatly increases the applicability of our approach, lowering the barrier to polyominoes optimization, much like grammatical evolution (GE) [67] did with regular languages.

Since our algorithm is greatly agnostic with respect to the genotypic representation employed by the EA, we compared different representations in terms of representation-related metrics (validity, redundancy, and locality). Moreover, to showcase the effectiveness of our approach, we evolved some polyominoes in a few case studies where the goal is to evolve a polyomino adhering to a grammar that is as much as possible similar to a pre-defined target polyomino. We showed experimentally that evolutionary optimization does work, giving polyominoes that are more and more similar to the target one while all adhering to the provided grammar, *i.e.*, meeting the user-defined constraints.

In this work, we extend our previous research presented in [51] in two ways. First, we analyze more in detail the relevant literature consisting the optimization of 2-D structures. Second, we consider a new case study where we apply our development algorithm to the evolution of a kind of modular soft robots, voxel-based soft robots (VSRs), whose physical structure can be described with a labeled polyomino. We consider the non-trivial task of the energy-efficient locomotion, a bi-objective optimization problem where we look for robots which run fast and energy-efficient at the same time and we compare two cases: one where we constrain the search to VSRs whose physical structure adhere 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 resorting on general-purpose EAs.

The remainder of the paper is structured as follows. In Section 2, we present the related works 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 connectivity, symmetry, and convexity. Connectivity refers to the number of holes within a polyomino, symmetry describes how a polyomino can be transformed through rotations and reflections, and convexity indicates whether the shape fully encloses a convex region. These properties provide a foundation for analyzing polyominoes across multiple contexts.

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, 9]. 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, 42].

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 [20]. 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 [41].

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 [77].

One prominent application is in percolation theory [8, 25], 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 [31], the spread of diseases [7], and information diffusion in networks [29]. 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 [75, 76].

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 [21, 57], the self-folding of three-dimensional (3-D) shells [63], and modular robotics [66]. In robotics, polyomino (2-D) and polycube (3-D) configurations enable adaptable, reconfigurable designs [30, 37, 40, 66], 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 [14].

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 one-dimensional (1-D) strings to 2-D arrays [17, 22]. 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 [14], where polyomino shapes emerge from the bonding of smaller tiles. Another approach, entitled Polyomino Development Algorithm, encodes an n-omino (polyminoes 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 [18]. 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 [59] 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 [52], 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 [24], by observing how clusters of cells evolve under different sets of rules.

Grammars, commonly used for 2-D pattern languages [22, 64], 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 connectivity (e.g., presence of holes), or convexity. Grammar-based methods have been applied to describe and enumerate various classes of polyominoes, especially convex polyominoes [12, 13, 19], and to support enumeration tasks using attribute grammars [10]. Beyond polyomino enumeration, spatial grammars have also been used in generative design of 3-D soft robots [11], 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 [26], successfully reproducing large datasets compiled from the literature. This representation has proven useful for reverse engineering of polymer design and production. 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 [51]. 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.

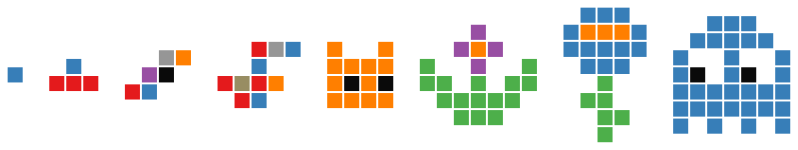


Fig. 1: Example of polyominoes with labels in $A = \{$🟥, 🟦, 🟩, 🟪, 🟧, ⬛, ⬜$\}$.

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. We denote by $\mathcal{P}_A$ 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).

Similarly to the case of CFGs for strings, we represent a PoCFG using a compact notation resembling Backus-Naur form (BNF), where rules are grouped by their non-terminal symbol, and the first rule corresponds to the starting symbol n_1. Figure 2 presents an example of PoCFG in BNF.

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 [68].

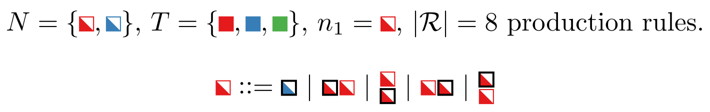

$N = \{$◩,◩$\}$, $T = \{$🟥,🟦,🟩$\}$, $n_1 =$ ◩, $|\mathcal{R}| = 8$ production rules.
::=

Fig. 2: Example of PoCFG. Half colored squares ◩ represent non-terminal symbols; fully colored squares 🟥 represent terminal symbols; on the right-hand-side, a thick black border ▣ denotes the reference cell in a referenced polyomino.

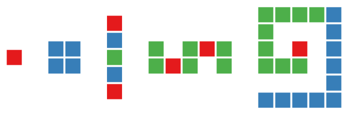


Fig. 3: Example of polyominoes belonging to $\mathcal{P}_{\mathcal{G}}$ with the PoCFG $\mathcal{G}$ of Figure 2.

3.3 Developing polyominoes given a PoCFG

In our previous work [51] 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 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, 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 process any $G \ni g$. We achieved this by making the algorithm modular. Specifically, we decoupled the components responsible for rule selection, rule expansion, and identification of suitable rules. 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 presents our development algorithm in the form of 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.

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($g, \mathcal{G}; c, o$)
2:      $p \leftarrow$ SINGLE(STARTINGSYMBOL($\mathcal{G}$))        ▷ init with starting symbol
3:      $s \leftarrow \varnothing$
4:      while true do
5:          $\{(x_i, y_i)\}_i \leftarrow$ NONTERMINALCELLS($p$)
6:          if $|\{(x_i, y_i)\}_i| = 0$ then        ▷ no non terminal cells
7:              break
8:          end if
9:          $(x^\star, y^\star) \leftarrow$ SELECTNONTERMINAL($\mathcal{R}; c$)        ▷ find cell to be replaced
10:         $\mathcal{R}_n \leftarrow$ OPTIONSFOR($p_{x^\star,y^\star}, \mathcal{G}$) ▷ find replacing ref. pol. for $p_{x^\star,y^\star} \in N$ in $\mathcal{G}$
11:         $(p', s) \leftarrow$ CHOOSEREPLACEMENT($\mathcal{R}_n, g, s$)  ▷ choose one replacing ref. pol.
12:         if $p' = \varnothing$ then        ▷ no chosen replacing polyomino
13:             return $\varnothing$
14:         end if
15:         if $\neg o \wedge \neg$FITS($p', p, x^\star, y^\star$) then        ▷ cannot replace $p$ with $p'$ at $x^\star, y^\star$
16:             return $\varnothing$
17:         end if
18:         $p \leftarrow$REPLACE($p, p', x^\star, y^\star$)
19:     end while
20:     return $p$
21: end function
```

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 $(x^\star, y^\star)$.

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() in Algorithm 1), selecting which one to expand is crucial, as it can impact on whether other steps fail due to production rules not being applicable. We treat this as a sorting problem and explore three sorting criteria for selection:

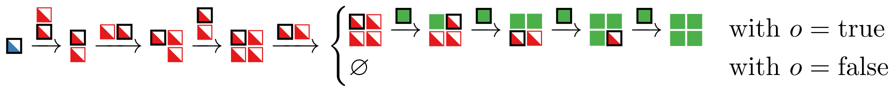


Fig. 4: An example of the development of a polyomino based on the PoCFG of Figure 2 with (o = true) or without (o = false) overwriting (with the Position criterion). Polyominoes in between arrows are the development stages: here the thick black border denotes the cell that is being replaced. Referenced polyominoes above arrows are the right-hand-side of the production rule being applied: here the thick black border denotes the reference cell.

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

Recency criterion: we choose the non-terminal cell most recently added to p (*i.e.*, at the most recent iteration of the algorithm) and, in case of tie, the one selected with the Position criterion; or

Free sides criterion: we choose the cell with the most free sides (*i.e.*, sides without adjacent cells), and, in case of tie, the one selected with 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 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) [38] 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 algorithms 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 it) 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 $\textsc{chooseReplacement}()$ 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 with the same g may yield different outputs. To formalize this, we include a state s as an argument for $\textsc{chooseReplacement}()$ 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.

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.*, $G = \{1, \dots, b\}^l \subseteq \mathbb{N}^l$, and the state s is an integer counter, *i.e.*, $s \in S = \{1, \dots, l\} \in \mathbb{N}$.

Given a genotype $g = (g_1, \dots, g_l)$, a set of production rules $\mathcal{R}_n = (r_1, \dots, r_k)$ 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 $\textsc{chooseReplacement}()$ 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, $\textsc{chooseReplacement}()$ 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 $S = \{1, \dots, l\} \in \mathbb{N}$.

Given $g = (g_1, \dots, g_l)$, a set of production rules $\mathcal{R}_n = (r_1, \dots, r_k)$ for the non-terminal n, and a state s, $\textsc{chooseReplacement}()$ works as follows. If the state, s, is $\varnothing$, 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, $g' = (g_s, g_{s+1}, \dots, g_{s+h})$ in g; (ii) it converts g' into an integer $z \in \{1, \dots, 2^h\}$; (iii) it selects the reference polyomino as $p' = p_j$, with $j = (z \bmod k) + 1$.

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

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$(8, 8)$.

$$g = \overbrace{\texttt{0111000100}\ldots\texttt{11}}^{(l=32)}$$

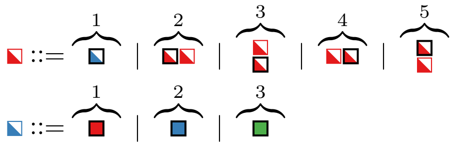
p	s	h	g'	z	k	j	p'
◪	1	3	011	3	5	4	◪◪
◪◪	4	3	100	5	5	1	◩
◩◪	7	2	01	1	3	2	🟦
🟦◪	9	3	001	1	5	2	◪◪
🟦◪◪	12	3	000	0	5	1	◩
🟦◩◪	15	2	11	3	3	1	🟥
🟦🟥◪	17	3	101	5	5	1	◩
🟦🟥◩	20	2	11	3	3	2	🟦
🟦🟥🟦	22						

(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 the development of a polyomino with the grammar of Figure 2 (also shown here in 5d for easing the comprehension and with the index of each rule made explicit) and three representations (one table for each representation). Each row in the table represents one iteration of the algorithm (with the Position sorting criterion and no overwriting). The thick black border denotes in p the cell that is being replaced and in p' the reference cell.

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 $S = \{1, \dots, l\} \in \mathbb{N}$.

Given $g = (g_1, \dots, g_l)$, 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 $j = \max(1, \lceil kg_s \rceil)$.

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, $\{1, \dots, |\mathcal{R}_n|\}$.

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 $N = \{n_1, \dots, n_m\}$ 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, $G = \{1, \dots, |\mathcal{R}_{n_1}|\}^{l_1} \times \dots \times \{1, \dots, |\mathcal{R}_{n_m}|\}^{l_m}$ and $S = \{1, \dots, l_1\} \times \dots \times \{1, \dots, l_m\}$, 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 rationale of this procedure is to have a number of genotype elements suitable for performing "enough" productions with each given non-terminal, while still constraining the genotype to be l-long. The parameter n_rec 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, $g = (g_{1,1}, \dots, g_{1,l_1}, \dots, g_{m,1}, \dots, g_{m,l_m})$, a set of rules $\mathcal{R}_{n_i}$ for a non-terminal n_i, and a state $s = (s_1, \dots, s_m)$ (or $s = \varnothing$), the function chooseReplacement() for this representation works as follows. If $s = \varnothing$, it initializes it as $s = (1, \dots, 1)$ (*i.e.*, a m-long vector of ones); otherwise, it updates the counter for the current non-terminal: if $s = (s_1, \dots, s_m)$, 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 the mod rule here is not needed, since the domain of each genotype part exactly matches the number of rules for the corresponding non-terminal symbol.

We denote this representation, which resembles the one of SGE [38], with 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 $\mathcal{G}$, 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, evolutionary strategy (ES) [27] or the more recent OpenAI-ES [69] are suitable for the reals(l) representation, while a genetic algorithm (GA) is appropriate for bits(l), possibly incorporating a linkage-exploitation mechanism [74] as in [46]. For simplicity, in this work, we use the simple GA described below, leaving the investigation of alternative EAs as future work. When experimenting with the modular soft robots (see Section 4.3.2), we use a bi-objective EA.

Given an objective function $f : \mathcal{P}_\mathcal{G} \to \mathbb{R}$ (assuming minimization problems, without loss of generality), we evolve polyominoes using a simple GA with two variation operators (mutation and crossover, each being representation-specific). The algorithm employs tournament selection for parents 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 representation-specific procedure. Then, for n_{gen} generations, we iterate through the following steps:

(1) We generate $r_{\text{x-over}} n_{\text{pop}}$ 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) We generate $(1 - r_{\text{x-over}}) n_{\text{pop}}$ new individuals via mutation, selecting the parent through tournament selection.
(3) We merge all newly generated individuals with the parents, hence obtaining a population P with $2n_{\text{pop}}$ individuals.
(4) We apply truncation selection to P, retaining the best n_{pop} individuals according to the objective function f.

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

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

As mutation, we use point-mutation, which randomly changes each genotype element to another value in the proper domain with p_mut probability, for $\text{ints}(l, b)$, $\text{bits}(l)$, and $\text{structured}(l, n_\text{rec})$. For $\text{reals}(l)$ we use the Gaussian mutation with standard deviation σ_mut.

As crossover, we use uniform crossover, which selects each element of the child genotype from one of the parents with equal probability, across all representations.

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 towards a predefined target shape while adhering to the given grammar, also in the more realistic case of evolutionary optimization of VSRs. Regarding the representations, we experimented with $\text{bits}(l)$, $\text{ints}(l, 4)$, $\text{ints}(l, 16)$, $\text{reals}(l)$, and $\text{structured}(l, 2)$, using different values for the genotype length l.

When comparing variants and representation, we focused on analyzing quantitatively some properties of the representation, since they allow to characterize how the search process operates [44, 65]. 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 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 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 matrices. We defined d_P as the Hamming distance between pair of polyominoes after having translated them in order to have coincident centers of mass. For d_G, we used the Hamming distance for $\text{bits}(l)$, $\text{ints}(l, b)$, and $\text{structured}(l, 2)$, and the Euclidean distance for $\text{reals}(l)$.

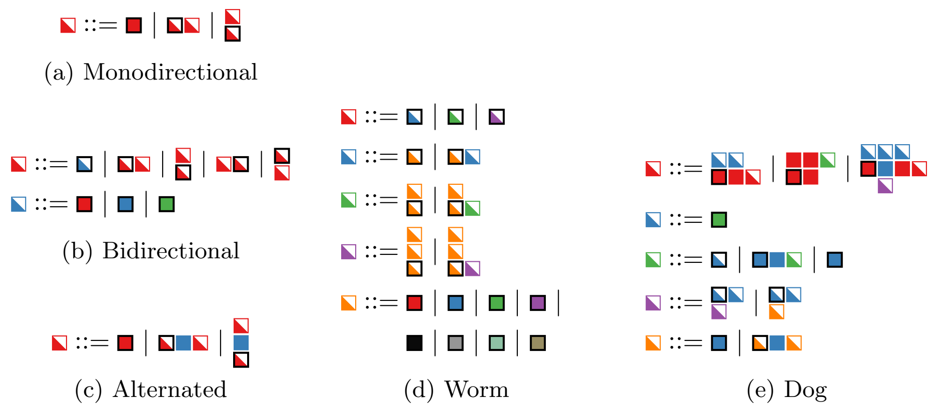


Fig. 6: The PoCFGs considered in the experiments. Half colored squares ◩ represent non-terminal symbols; fully colored squares 🟥 represent terminal symbols; on the right-hand-side, a thick black border ▣ denotes the reference cell in a referenced polyomino. e.g., for the Dog grammar, $N = \{$◩, ◩, ◩, ◩, ◩$\}$, $T = \{$🟥, 🟦, 🟩$\}$, $n_1 =$ ◩, and there are $|\mathcal{R}| = 11$ production rules.

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 $l \in \{10, 15, \dots, 245, 250\}$. We conducted similar experiments for the other representations and observed qualitatively similar findings. For each l value, we generated 5000 genotypes (and their corresponding phenotypes) to measure validity and uniqueness, and 1000 genotypes to measure locality. 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). 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 [28, 39, 55]. 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

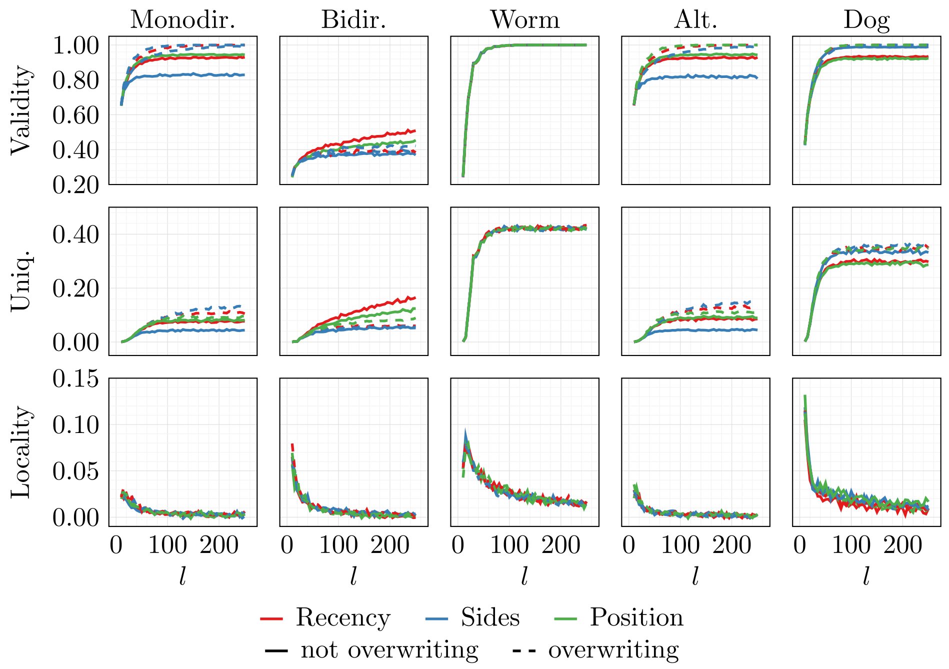


Fig. 7: Representation properties (rows of plots) for the six variants of the development algorithm (line colors and types) measured on the five grammars (columns of plots) with the bits(l) representation.

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 clear and general distinctions can be made 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 in the corresponding phenotypes. This interplay between locality and genotype usage has been previously observed and can be analyzed through visualization tools [50].

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.

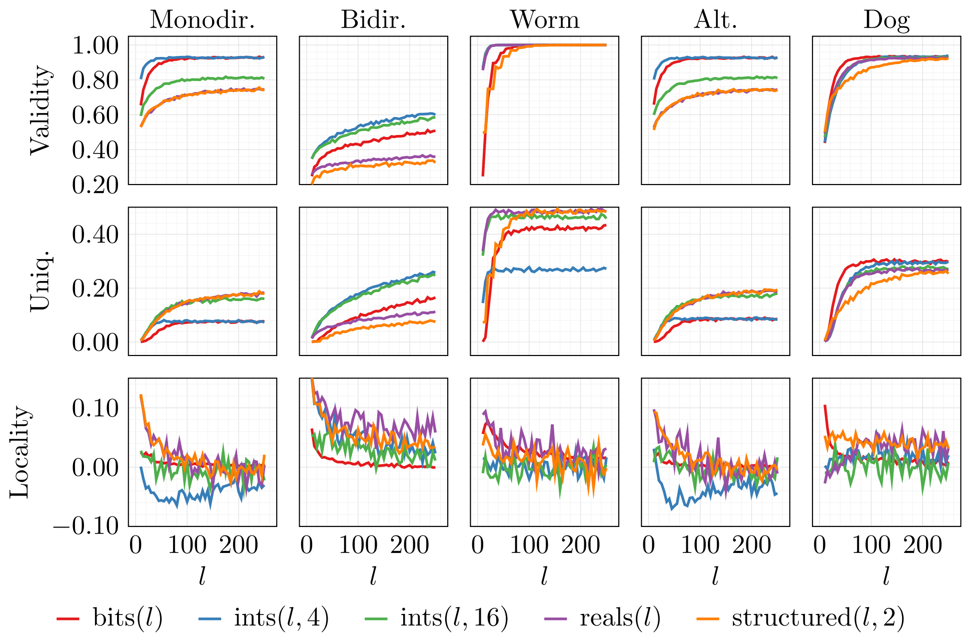


Fig. 8: Representation properties (rows of plots) for the five representations (line color) measured on the five grammars (columns of plots) with the development algorithm based on Recency without overwriting.

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: 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).

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

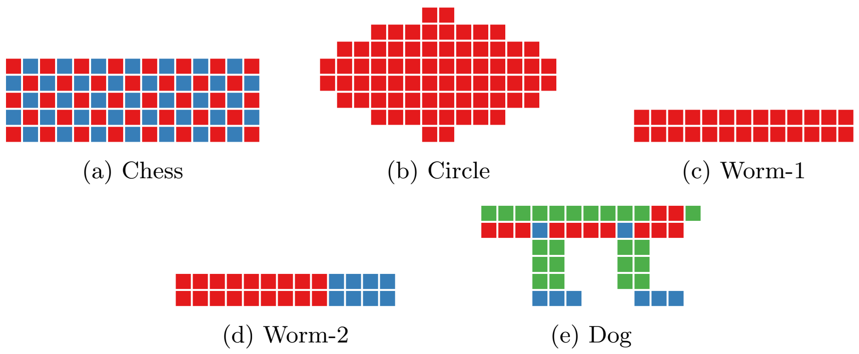


Fig. 9: The five target polyominoes.

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.

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 purposely chose target polyominoes which match very differently the five PoCFGs. Note that the Dog shape is not perfectly achievable with the Dog grammar, due to the misplaced rightmost foot.

We evolved the polyominoes using the GA described in Section 3.4 with the following parameters (summarized in Table 1): $n_{\text{pop}} = 100$, $n_{\text{gen}} = 200$, $r_{\text{x-over}} = 0.8$, $n_{\text{tour}} = 3$, $p_{\text{mut}} = 0.01$, the latter being the only representation-specific parameter. We employed the bits(500) representation with the Recency criterion and no overwriting: 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.

We used JGEA [48] for the experiments. For each of the 25 combinations of grammar and target, we performed 50 evolutionary runs, varying the random seed.

Parameter	Value
Number of generations n_{gen}	200
Population size n_{pop}	100
Tournament size n_{tour}	3
Crossover rate $r_{\text{x-over}}$	0.80
Mutation probability p_{mut}	0.10

Table 1: Our EA parameters.

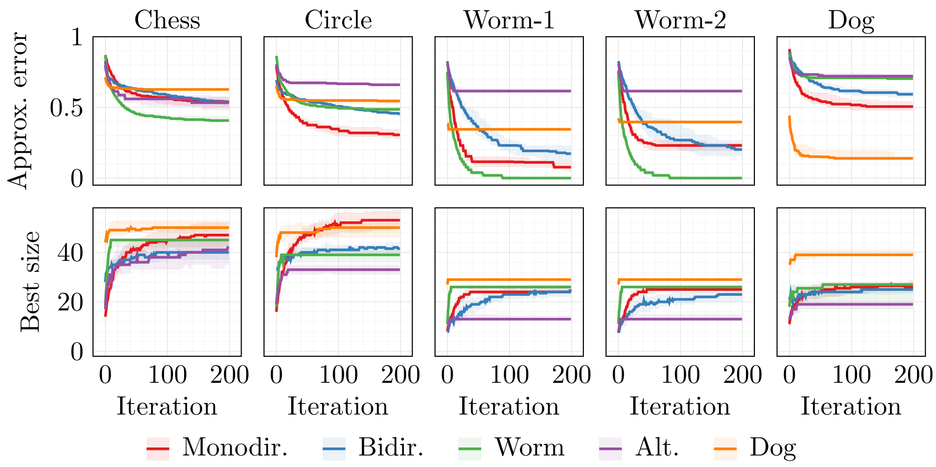
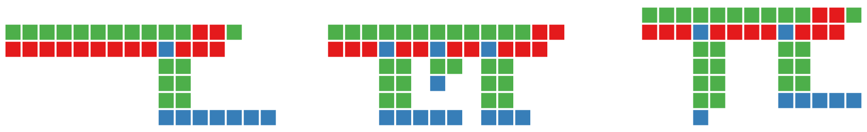


Fig. 10: Approximation error and size (row of plots) of the best polyomino during the evolution for the five problems (column of plots) using the five grammars (color line) with the development algorithm based on Recency without overwriting and the bits(500) representation. The shaded area corresponds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

Results and discussion

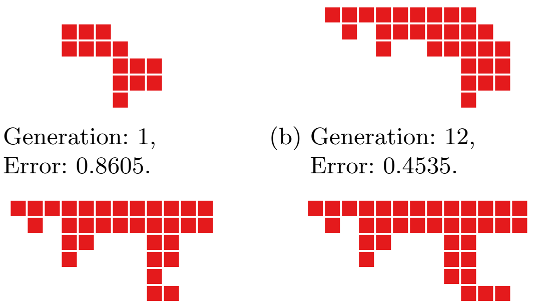
Figure 10 presents the results of these experiments, showing both the approximation error of the best polyomino during the evolution and its size, measured by the number of cells.

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 other grammars in solving the Dog problem. This highlights the significance of a well-designed grammar, not only for enforcing structural constraints, but also to incorporate domain-specific knowledge about the problem. Figure 11 illustrates the evolution of the best individual across generations when using the Dog grammar to evolve the Dog shape, for one random run. In contrast, Figure 12 presents an example of evolution using the Monodirectional grammar for the same target and one random run. While the resulting shape bears some resemblance to the desired form, the lack of necessary label colors (terminal



(a) Generation: 1, Error: 0.3605.
(b) Generation: 7, Error: 0.31.
(c) Generation: 116, Error: 0.1111.

Fig. 11: Snapshots on different generations of the best individual evolved to match the polyomino target Dog (Figure 9e), using the Dog grammar (Figure 6e).



(a) Generation: 1, Error: 0.8605.
(b) Generation: 12, Error: 0.4535.
(c) Generation: 26, Error: 0.4535.
(d) Generation: 102, Error: 0.3605.

Fig. 12: Snapshots on different generations of the best individual evolved to match the polyomino target Dog (Figure 9e), using the Monodirectional grammar (Figure 6a).

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 employing 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 tended to get trapped in local minima after a few iterations. This is indicated by the size of the fittest individual, which remains unchanged, highlighting the EA difficulty to explore the solution space effectively under these circumstances.

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.

4.3.2 Evolving simulated modular soft robots

In order to further test the capability of our approach to evolve 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 anticipated 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, 45], rather than with their 3-D counterpart, which can be actually fabricated [30, 37], but whose body is not a 2-D polyomino.

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 (*voxels*), 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 [45]. 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 [15, 16], 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.*, $\rho_N^{(k)} = \rho_E^{(k)} = \rho_S^{(k)} = \rho_W^{(k)} = 1 + \rho_{\max} \sin(2\pi k \delta t)$;

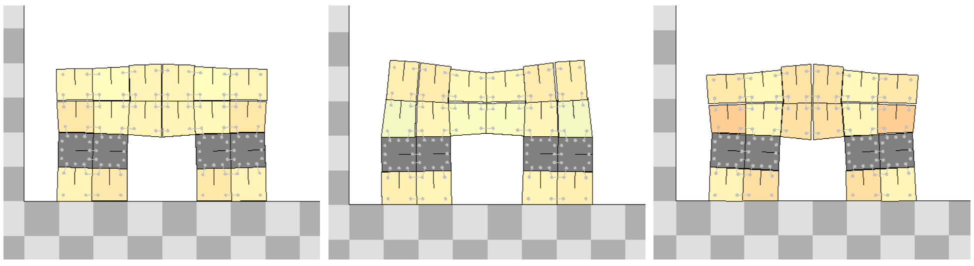

(a) $t = 1.0\,\mathrm{s}$ (b) $t = 1.6\,\mathrm{s}$ (c) $t = 2.2\,\mathrm{s}$

Fig. 13: 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 ar PS (▪▪) modules. That is, ▦. 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 voxels. The gray object with a checkerboard pattern is the ground, where the VSR is staying.

- 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 [49] 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 13 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 not 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 [60, 61]. 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 [72]. Recently, some symbolic artifacts, as, e.g., graphs or ensembles of regression trees have also been explored, possibly for favoring transparency [47, 54]. 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 [53].

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 $t = 30\,\mathrm{s}$.

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:

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

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.

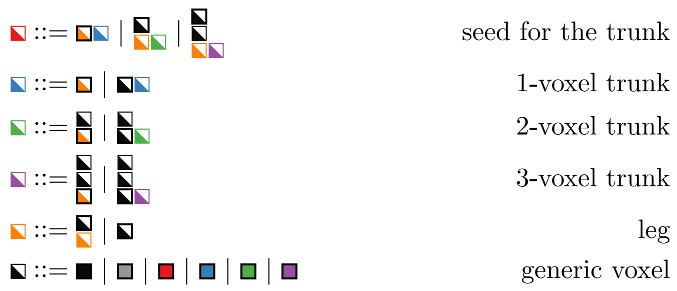
We looked for VSRs which were both fast and energy efficient, *i.e.*, we considered a bi-objective optimization problem consisting in maximizing v_x and minimizing p. Clearly, there is a trade-off between the two objectives: in particular, a VSR composed only of passive voxels (PH or PS) scores perfect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

Evolutionary algorithm (EA). We used the non-dominated sorting genetic algorithm II (NSGA-II) for tackling the bi-objective optimization problem of the efficient locomotion of VSRs. We remark that the approach proposed in this work is agnostic with respect to the EA being used for the optimization in the space of polyominoes. Indeed, with this experiment we also wanted to validate this claim.

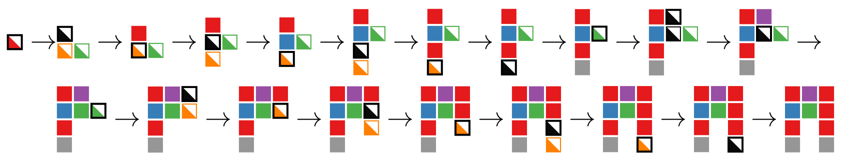
We used the implementation of NSGA-II available in JGEA with default parameters. Namely, we set it to use a population of 100 individuals and to build the offspring using crossover for 80 % of the new individuals and mutation for the remaining ones. As we dealt with a discrete genotype space (see next section), we employed a simple diversity promotion mechanism: whenever we generated a new genotype (either during the initialization or in the reproduction phase), we discarded it and generated a new one if it was already present in the population.

Representation. We employed a representation of VSRs based on our approach. In particular, we (a) designed a PoCFG $\mathcal{G}_{\mathrm{VSR}}$ for describing a set of VSRs where to search for and (b) we chose the bits(1024) representation (see Section 3.3.3), *i.e.*, we used the genotype space $G = \{\mathtt{0}, \mathtt{1}\}^{1024}$, with the Recency criterion and no overwriting (see Section 3.3.2).

We show $\mathcal{G}_{\mathrm{VSR}}$ in Figure 14a. It has six terminal symbols ($T = \{$⬛, ◼, 🟥, 🟦, 🟩, 🟪$\}$, corresponding to PH, PS, AH, AV, AS, AC voxels, respectively), six non-terminal symbols ($N = \{$◩, ◩, ◩, ◩, ◩, ◩$\}$), and 17 production rules. Intuitively, $\mathcal{G}_{\mathrm{VSR}}$ describes



(a) The PoCFG $\mathcal{G}_{\text{VSR}}$ for describing biped-like VSRs.



(b) An example of the development of a VSR adhering to $\mathcal{G}_{\text{VSR}}$.

Fig. 14: The PoCFG $\mathcal{G}_{\text{VSR}}$ for biped-like polyominoes, used in our experiments with VSRs, and an example of development of a polyomino adhering to $\mathcal{G}_{\text{VSR}}$ (with the Recency sorting criterion and no overwriting). In 14a, the thick black border denotes the reference cell. In 14b, the thick black border denotes the cell being replaced.

the subset of VSRs whose morphology resemble a biped, *i.e.*, a trunk with two legs. In particular, the horizontal trunk can be one, two, or three voxels wide and with a unbounded length; each 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. Voxels of any type can be used in every part of the morphology. In Figure 14b we show an example of the development (with the Recency criterion and no overwriting) of a polyomino adhering to $\mathcal{G}_{\text{VSR}}$.

As a comparison baseline, we also employed another representation of 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 a int string $g \in G$, we first reshape it to a 6×4 matrix, then we consider the larger polyomino in the matrix formed by non-zero elements, and finally we map each matrix element greater than zero to a corresponding value in {⬛, ◼, 🟥, 🟦, 🟩, 🟪}.

Summarizing, we compared two approaches for tackling the efficient locomotion optimization problem. Both use NSGA-II as EA and search in the space of (a subset of) VSRs. The $\mathcal{G}_{\text{VSR}}$-based approach uses $\{\texttt{0}, \texttt{1}\}^{1024}$ as genotype space; the grid-based approach uses $\{0, 1, 2, 3, 4, 5, 6\}^{24}$ as genotype space. For both, we used the uniform

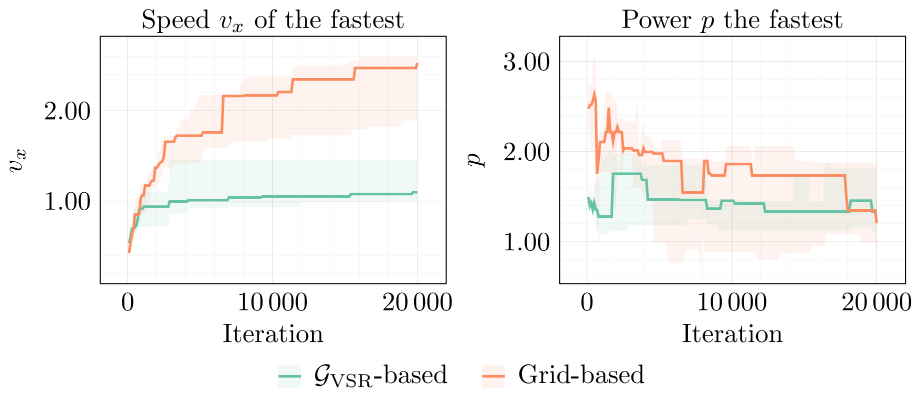


Fig. 15: Average speed v_x (right) and power p (left) of the fastest robot in the population during the evolution, for the two approaches. The shaded area corresponds to the interquartile range, the line to the median across 10 runs.

crossover and the point-mutation as genetic operators and the random initialization in the domain of each genotype element for population initialization.

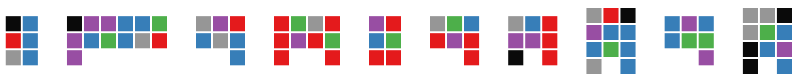
Results and discussion

We executed 10 evolutionary runs for each of the two approaches. We used JGEA for the optimization and 2D-VSR-SIM [45] for simulating the VSRs. We set $n_{\text{pop}} = 100$, $p_{\text{mut}} = \frac{1}{|g|}$ (*i.e.*, $\frac{1}{1024}$ for the $\mathcal{G}_{\text{VSR}}$-based approach and $\frac{1}{24}$ for the grid-based one), and stopped each run after 20 000 fitness evaluations.

Figure 15 shows the progression of the speed v_x and power p of the fastest VSR in the population during the evolution for the two representations. We remark that we considered a bi-objective optimization problem: however, we arbitrarly 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 15 that VSRs evolved with the grammar-based representation were in general slower, but more efficient, at least until the last stage of the evolution, when grid-based VSRs tended to become equally efficient. This difference can be explained in two ways. First, the subsets of VSRs in which the EA was searching with the two representations corresponded to different trade-offs between speed and power: namely, bipeds-like VSRs are likely slower than “free-form” VSRs. Second, the different directedness of the genotype-phenotype mapping induced different fitness landscapes which were more or less convenient to navigate for the EA. We do not have strong arguments for none of the two hypotheses. In particular, concerning the former, 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 [73].

What really matters for our study is that with the $\mathcal{G}_{\text{VSR}}$-based representation, 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 16


(a) $\mathcal{G}_{\text{VSR}}$-based best VSRs.

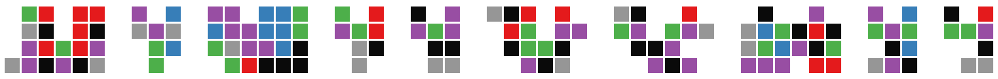

(b) Grid-based best VSRs.

Fig. 16: The fastest VSR obtained in each one of the 10 evolutionary runs for the two approaches.

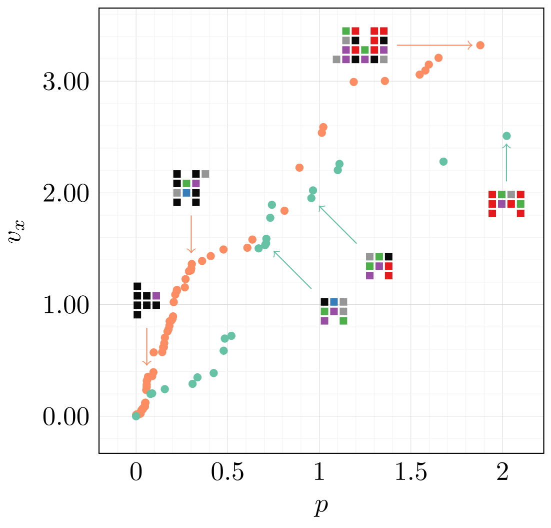
the fastest VSR obtained in each of the 10 repetitions for the two approaches. It can be easily noted that all the VSRs evolved with the $\mathcal{G}_{\text{VSR}}$-based are actually bipeds—some of them with zero-long legs, some with short trunks. Conversely, robots evolved with the grid-based approach are actually free-form (though always of size of at most 6×4). In order to apply the biped-like constraint to the grid-based case, 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 fitness, or maybe a third objective. Arguably, the effort would have been greater than the one required to define $\mathcal{G}_{\text{VSR}}$.

Finally, to further remark the effects of constraining the search using a PoCFG, we show in Figure 17 the values of the objectives for each VSR in the population for the two best repetitions of the two approaches, *i.e.*, the ones giving the fastest VSR at the end of the evolution. For a few VSRs for each approach, we also show the corresponding morphology.

It can be seen that the two representations were apparently differently able to cover different regions of the Pareto frontier, *i.e.*, different v_x-p trade-offs. In particular, the $\mathcal{G}_{\text{VSR}}$-based representation delivered more robots in the “middle”, while the grid-based representation delivered more faster and efficient robots. Moreover, by “counting” the markers for the two colors, it can be seen that the grid-based representation apparently produced more VSRs. As the two representations were used with the same EA (and the same population size of $n_{\text{pop}} = 100$), this means that the $\mathcal{G}_{\text{VSR}}$-based representation lead to a less phenotypically diverse population. We recall that (a) we enforced genotype diversity in both cases at the level of the EA and (b) the two sets of polyominoes corresponding to the two representations are differently large. The grid-based set size is $< 7^2 4$ while the grammar-based one is infinite. The tendency of grammar-based representations to suffer from poor phenotype diversity is not new and is related to degeneracy and redundancy [44, 50]. More broadly, we believe it is a price to pay for the possibility of conveniently restrict the search space through a grammar.

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,



— $\mathcal{G}_{\text{VSR}}$-based — grid-based

Fig. 17: Average speed v_x and power p of each VSR in the final population of the best run for the two approaches (*i.e.*, the one delivering the VSR with the greatest v_x among the 10 evolutionary runs). For three VSRs for each approach, we show the morphology.

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 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 an EA in the context of evolutionary optimization. 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 [32] or DNA shapes [70].

Acknowledgements. This research is the result of the collaboration with the Department of Engineering and Architecture of the University of Trieste, Italy; supported by the 2023 SPECIES scholarship. The first author is funded by FCT - Foundation for Science and Technology, under the grant 2022.10174.BD. This work was supported by the Portuguese Recovery and Resilience Plan (PRR) through project C645008882-00000055, Center for Responsible AI, by the FCT, I.P./MCTES through national funds (PIDDAC), by Project No. 7059 - Neuraspace - AI fights Space Debris, reference C644877546-00000020, supported by the RRP - Recovery and Resilience Plan and the European Next Generation EU Funds, following Notice No. 02/C05-i01/2022, Component 5 - Capitalization and Business Innovation - Mobilizing Agendas for Business Innovation, and by national funds by FCT, I.P., in the framework of the Project UIDB/00326/2025 and UIDP/00326/2025.

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 (2006)

[3] Barequet, G., Ben-Shachar, G.: Counting polyominoes, revisited (May 2024)

[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] Conway, A.: Enumerating 2d percolation series by the finite-lattice method: Theory. Journal of Physics A: Mathematical and General **28**(2), 335 (1995)

[9] Delest, M.: Polyominoes and animals: some recent results. Journal of mathematical chemistry **8**(1), 3–18 (1991)

[10] 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

[11] 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

[12] 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

[13] 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

[14] 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)

[15] 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)

[16] 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)

[17] Fernau, H., Schmid, M.L., Subramanian, K.G.: Two-dimensional pattern languages. In: Workshop on Non-Classical Models for Automata and Applications (2017)

[18] 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

[19] Frosini, A., Rinaldi, S., et al.: An object grammar for the class of l-convex polyominoes **17**(1-2), 97–110 (2006)

[20] 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

[21] 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

[22] Giammarresi, D., Restivo, A.: Two-Dimensional Languages, pp. 215–267. Springer Berlin Heidelberg, Berlin, Heidelberg (1997), ISBN 978-3-642-59126-6

[23] 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)

[24] 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)

[25] Grimmett, G.: What is Percolation?, pp. 1–31. Springer Berlin Heidelberg, Berlin, Heidelberg (1999), ISBN 978-3-662-03981-6

[26] 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

[27] Hansen, N., Arnold, D.V., Auger, A.: Evolution strategies. Springer handbook of computational intelligence pp. 871–898 (2015)

[28] Harper, R.: GE, explosive grammars and the lasting legacy of bad initialisation. In: IEEE Congress on Evolutionary Computation, IEEE (Jul 2010)

[29] 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

[30] Hiller, J., Lipson, H.: Automatic design and manufacture of soft robots. IEEE Transactions on Robotics **28**(2), 457–466 (2011)

[31] 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

[32] 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)

[33] Knight, T., Stiny, G.: Making grammars: from computing with shapes to computing with things. Design Studies **41**, 8–28 (2015)

[34] Knuth, D.E.: Dancing links. arXiv preprint cs/0011047 (2000)

[35] 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

[36] 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)

[37] 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)

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

[39] 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)

[40] 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

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

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

[43] 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

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

[45] 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)

[46] 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)

[47] 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

[48] 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)

[49] 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)

[50] 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)

[51] 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

[52] Mordvintsev, A., Randazzo, E., Niklasson, E., Levin, M.: Growing neural cellular automata. Distill **5**(2), e23 (2020)

[53] 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)

[54] 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)

[55] Nicolau, M., Agapitos, A.: Understanding grammatical evolution: Grammar design. In: Handbook of Grammatical Evolution, pp. 23–53, Springer International Publishing (2018)

[56] 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)

[57] 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

[58] Ota, P.A.: Mosaic grammars. Pattern recognition **7**(1-2), 61–65 (1975)

[59] Packard, N.H., Wolfram, S.: Two-dimensional cellular automata. Journal of Statistical Physics **38**(5), 901–946 (1985)

[60] 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)

[61] Pellegrino, F.A., Blanchini, F., Fenu, G., Salvato, E.: Data-driven dynamic relatively optimal control. European Journal of Control **74**, 100839 (2023)

[62] 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)

[63] 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

[64] Prusa, D., Hlavá, V.: 2d context-free grammars: Mathematical formulae recognition. In: Prague Stringology Conference (2006)

[65] Rothlauf, F., Goldberg, D.E.: Redundant representations in evolutionary computation. Evolutionary Computation **11**(4), 381–415 (2003)

[66] 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)

[67] 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)

[68] Sakai, I.: Syntax in universal translation. In: Proceedings of the International Conference on Machine Translation and Applied Language Analysis (1961)

[69] Salimans, T., Ho, J., Chen, X., Sidor, S., Sutskever, I.: Evolution strategies as a scalable alternative to reinforcement learning. arXiv preprint arXiv:1703.03864 (2017)

[70] 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)

[71] Subramanian, K., Ali, R.M., Geethalakshmi, M., Nagar, A.K.: Pure 2d picture grammars and languages. Discrete Applied Mathematics **157**(16), 3401–3411 (2009)

[72] 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)

[73] 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)

[74] 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)

[75] Vanderzande, C.: Lattice Models of Polymers. Cambridge Lecture Notes in Physics, Cambridge University Press (1998)

[76] 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

[77] Whittington, S.G., Soteros, C.E.: Lattice animals: Rigorous results and wild guesses (1990)

[78] Winslow, A.: Staged self-assembly and polyomino context-free grammars. Natural Computing **14**(2), 293–302 (2015)