



Using estimation of distribution algorithm for procedural content generation in video games

Arash Moradi Karkaj¹ · Shahriar Lotfi² 

Received: 28 June 2021 / Revised: 17 April 2022 / Accepted: 2 June 2022 /
Published online: 2 August 2022

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2022

Abstract

Content generation is one of the major challenges in the modern age. The video game industry is no exception and the ever-increasing demand for bigger titles containing vast volumes of content has become one of the vital challenges for the content generation domain. Conventional game development as a human product is not cost efficient and the need for more intelligent, advanced and procedural methods is evident in this field. In a sense, procedural content generation (PCG) is a Non-deterministic Polynomial-Hard optimization problem in which specific metrics should be optimized. In this paper, we use the Estimation of Distribution Algorithm (EDA) to optimize the task of PCG in digital video games. EDA is an evolutionary stochastic optimization method and the introduction of probabilistic modeling as one of the main features of EDA into this problem domain is a reliable way to mathematically apply human knowledge to the challenging field of content generation. Acceptable performance of the proposed method is reflected in the results, which can inform the academia of PCG and contribute to the game industry.

Keywords Computer games · Procedural content generation · Estimation of distribution algorithm · Univariate marginal distribution algorithm and probabilistic modeling

Handling Editor: Sebastian Risi.

✉ Shahriar Lotfi
shahriar_lotfi@tabrizu.ac.ir

Arash Moradi Karkaj
A.MoradiEDU@gmail.com

¹ Department of Computer Engineering, University College of Nabi Akram, Tabriz, Iran

² Department of Computer Science, Faculty of Mathematics, Statistics and Computer Science, University of Tabriz, Tabriz, Iran

1 Introduction

Procedural Content Generation (PCG) in digital games is an active research field, but the performance of many evolutionary methods is yet to be tested in this domain. Specifically in the domain of digital computer games, which are mainly a media for entertainment and consist of various genres that require immense virtual environments with extensive content volume. This task demands the possibility of extracting data from the problem space while generalizing human knowledge into the core of the evolutionary process, which highlights the importance of finding an efficient approach. PCG refers to the process of generating content using certain algorithms or procedures, often used in tandem with Player Experience Modeling (PEM), which is a modeling approach based on interactions of the players with the game during gameplay [1]. Moreover, the evolutionary algorithms are often used in PCG. These methods are a subsection of evolutionary computation and often categorized as population-based meta-heuristic optimization approaches. The perceived concept of PCG as an Optimization problem consists of a big solution space in which the candidate solutions are selected and bred using efficient and intelligent methods [2].

PCG in video games can be formulated as a Non-deterministic Polynomial-Hard (NP-Hard) problem, similar to the famous knapsack problem, in which there are several criteria to be met and the overall performance needs to be optimal, best defined in the field of combinatorial optimization. Through this lens, PCG for digital games requires incorporating an artificially intelligent approach. Most of the research in this field uses a combination of various Artificial Intelligence (AI) methods that often include additional constraints along with evolutionary methods in order to keep the population acceptable and playable while having heuristics to guide the process effectively. In platformer games, expressivity metrics like linearity, density and leniency of the challenges provided along with pattern quantity are also applied to ensure the generated content's quality.

In this paper, we propose the use of the Estimation of Distribution Algorithm (EDA). EDA is a stochastic optimization method that works towards finding the global optima by building, sampling and updating comprehensive probabilistic models of the problem space. Controllability and high-level computational modeling based on problem structure are the two useful features that make this method ideal for PCG in video games.

The main objective of PCG using EDA is to optimize the process of generating content (game levels) using the expressivity metrics. Considering the excellent performance of EDA in solving similar complex problems and utilizing its unique advantages, we aim to achieve a better content generation process while successfully meeting the required expressivity metrics, comprehensively studied by Horn et al. [3]. In this paper, we set out to find whether EDA can optimize PCG in video games while satisfying certain criteria.

The Outline of this paper is as follows: In Sect. 2, we provide background and related work. Section 3 introduces the proposed method. In Sect. 4, experimental results are provided. Finally, Sect. 5 contains conclusion and future work.

2 Background and related work

In this section, first, we point out some basic definitions and theoretical foundations related to the PCG research field, which concludes with a compact classification of various approaches. Next, we begin to review the relevant and major studies in this domain.

2.1 Background

The following section discusses important definitions, classifications and principles used in related studies. Starting with platformer games, which are a well-established genre of games in which the player traverses through a game level, overcoming several obstacles and reaching the end. Super Mario Bros. (SMB) (Nintendo, 1985) is considered to be one of the prominent titles in this genre, giving rise to various tested frameworks used in game research, i.e. PCG.

PCG in video games was first introduced in the 1980's. Pioneers of this approach are games like *Rogue* and *NetHack*, while more recent entries in the realm include *No Man's Sky* and *Dwarf Fortress*. PCG methods branch out to either constructive or generate and test and search-based PCG is a certain type of generate-and-test method with a major difference: the evaluation method in this approach does not eliminate or approve solutions but simply scores them based on the defined objective function. The similarity between this evolutionary technique and search-based content generation makes it ideal for researchers [4, 5]. Genetic algorithm is the next widely used approach in this field. Methods like Answer Set Programming (ASP) [6] and Grammatical Evolution [7] have also gained much recognition and acceptable results. In addition, EDA methods such as CMA-ES in combination with the Quality diversity (QD) approaches have also been introduced to this domain [8].

However, the quality of the generated content is hugely dependent on the preferences of the player whom experiences it. This principle gives rise to another field of studies know as Experience-Driven Procedural Content Generation (EDPCG). The main aim in this field is to build precise and efficient computational models that make content generation suitable to the expectations of each player, often dubbed Player Experience Modeling (PEM). In turn, PEM acts as the objective function to evaluate the content generated in the evolutionary methods [1].

2.2 Related work

PCG in video games was first introduced in the 1980's. However, in 2004, the initial attempts to capture the player experience model and apply player behavioural data made its academic debut. Later in 2006, with the first studies in PCG methods, the latter and PEM studies became complementary.

With the aim of increasing game compatibility with player tendencies, skill level and expectations, Charles et al. [9] introduce the first academic study in the field of PEM. In this study, in order to tailor content to players' satisfaction, the interests

and skill level of the player are modelled, evaluated and optimized. While this study has novel ideas, the proposed model does not include any actual implementation, as it is mostly theoretical, simply pointing out the framework and an overview of the technique. Yannakakis et al. attempt to rectify this impracticality of PEM in series of research in the years of 2004 [10], 2005 [11] and 2007 [12] with the first steps towards adaptation in the game Pac-Man (T. Iwatani, 1980). In the first study, the aim is to make the game session more fun by online learning of player behaviour and corresponding adaption of enemy behaviour. In the next step, previous results are optimized by using PEM and applying Bayesian Networks. Ultimately, they set out to validate previous research by including human players and measuring their reactions. The goal in this study is to test the accuracy of the satisfaction measurement technique and it is the first research that verifies the output of a method with a human participant. While this study provides valuable insight and good results, the underlying game is simple and limited.

Yannakakis et al. [13] present a technique to improve fun in virtual reality movement games. The main agenda in this study is to apply the gradient ascent technique to player models to adjust parameter allocation and thus, increase entertainment value. Modeling in this study uses Sequential Forward Selector (SFS) algorithm for feature selection and machine learning techniques. This approach has great generalization value to other methods, but on the other hand, the noise level is considerably high which reduces the combinational model's accuracy. This deficiency is addressed in their later work. In the new approach, Large Margin Algorithm (LMA) and Gaussian techniques (Bayesian and linear learning) derived from other studies are combined with meta-LMA and neural-evolutionary (non-linear) methods. This unique set of techniques, together with comprehensive SFS and nBest individual feature selection algorithms, results in efficient modeling. To sum up, this method produces suitable results using actual data, but has limited implementation and low generalization potential [14].

Pedersen et al. [15] first attempt to find a relation between design parameters in platformer games and various player experiences and play styles. The method used here is preference learning that models the player experience by using neural networks and applies nBest, SFS and Perceptron Feature Selection (PFS) techniques to establish an efficient feature selection module. Single-layer neural networks and a minimal number of selected features provide good performance, but the overall process is highly human-dependent and very limited. Next, they try to predict player's response to a challenge using intelligent computational methods in computational modeling [16]. While the performance report of multi-layer neural networks and a variety of search algorithms for feature selection are among the major contribution of these studies, on the other hand, feature selection limits the modeling and human survey increases human interaction. In a similar effort, Shaker et al. [17] attempt to collect game session data to model and predict player reaction to game challenges. They use the concept of sequential features, which is used in the Generalized Sequential Pattern (GSP) algorithm. Defining an optimized length for such sequential features result in increased performance.

Togelius et al. [18] introduce the first academic study in the realm of PCG. The main objective of this study is to maximize the player satisfaction in racing games

with automatically generated racetracks. PEM and several evolutionary methods are used to this end, and the objective function for evaluating the tracks is based on generic criteria related to player satisfaction. In order to further augment of PEM, Togelius et al. [5] attempt to increase their modeling robustness. The key enhancement in this study is feature selection, which enables better player skill detection and racing style classification. A pre-configured controller module is given a default driving ability, but with each race, it samples the player behaviour, which in turn, provides better results while enhancing the player modeling. However, the pursuit for a more detailed PEM continues and Togelius et al. [19] include a more comprehensive objective function derived from Malone [20] theory of fun, while applying multi-layer neural networks that improve feature selection functionality. Great flexibility in using evolutionary techniques and search-based approaches, impressive controllability in the content generation process and generalization are some of the main achievements of these efforts. Meanwhile, the study is mainly experimental and does not produce any practical results.

Smith et al. [21] use the concept of “rhythm” as their principle in level generation. In this method, both the rhythm and the levels are generated procedurally, but a human designer controls the vast majority of level design process. The Launchpad content generator introduced in this study is grammar-based and operates in two stages. First, the overall rhythm is constructed by placing different phrases. Next, based on this rhythm, using predetermined Design Element (DE) of level segments, the final content is generated. The importance of this study is twofold, first in its introduction of the “expressive range” as an analysis tool in the PCG domain that enables comparison between various methods’ outputs, and later developments in 2010 [22], which provides a comprehensive set of criteria for meaningful comparison. Second, the rationale of rhythm is the source of further developments in devising an evaluation framework, which ultimately generalizes to other studies. However, the content generator itself remains the same and only focuses on satisfying linearity and leniency metrics of the levels. Later, Jennings-Teats et al. [23] follow this research direction with the aim of adding adaptable difficulty. Difficulty adaptation in this approach is dynamic, online and real-time, meaning future content is based on the player’s skill level. The online implementation and real-time computations are efficient, but the dependency on the data derived from human players is very limiting. On the other hand, the overall structure of the system is simplistic and is only applicable on platformer games.

Sorenson et al. [24–26] propose a top-down approach to eliminate the limitations and lack of flexibility of bottom-up methods based on the Feasible-Infeasible TwoPopulation (FI-2Pop) genetic algorithm, while drawing on the rationale of “rhythm groups” as first introduced in previous studies [21]. The main objective of this study is to provide a coding scheme that can be generalized to other content generation systems with as much alteration as possible. Moreover, unlike other methods, player satisfaction and amount of fun is not a by-product of the system and is monitored closely via a direct objective function. This objective function is based on the renowned theory of “flow” by Csikszentmihalyi [27], and other theories of “psychology of fun.” The inclusion of the theory of flow reinvents the task of adaptive difficulty in PCG methods and accounts for the main contribution of this

strand of studies. Moreover, efficient genotype coding and the top-down approach efficiency in detecting variety in level design are noteworthy, but the generated levels are too simplistic and repetitive and the human intrusion makes the system less automatic and intelligent on its own.

Shaker et al. [28] introduce an online adaptation mechanism for maximizing player entertainment. This method uses neural networks for PEM, while exclusively applying SFS algorithm for feature selection, which makes it unique from other approaches [13, 14]. Online computations and modeling have low computational load, but the resulting player models are not completely coherent with actual player experience. To remedy this deficiency, they focus on defining a suitable set of analytical features via limiting the feature selection frame. The idea of gradual level division and using controllable features proves useful, but the overall mechanism adds to human interference and makes the approach semi-procedural [29]. In order to increase the performance of previous PCG methods, they use the concept of Grammatical Evolution. The secondary objective of this study is to provide a comprehensive framework to analyze the expressive range of various methods and enable comparison [7]. Efficient combination of grammatical and evolutionary approaches, providing top-down view of PCG and enabling more control over the process are the main advantages of this method. However, indirect mapping from phenotype to genotype results in extra computational load on the online method.

Smith et al. [6] use answer set programming to model the design space using the logic programming, then the modeled problem is solved using answer sets and interpreted into the game content. This approach has a high-level view of the task, works efficiently and produces suitable results when combined with other bottom-up methods, but has limited control over the content due to indirect encoding and high-level structure.

Dahlskog et al. [30] propose the Pattern Based (PB) method. In this definition, the generated levels consist of three patterns: (1) micro-patterns (small patterns extracted from the original game), (2) meso-patterns (objective of the optimization algorithm to maximize the count), that create the complete level that is called (3) macro-patterns. High generalizability is the main advantage of this approach, but the results are too similar to the original game. In order to remedy some of the shortcomings of the search-based techniques, Roberts et al. [31] attempt to control the content generation system with machine learning principles. While this approach eliminates the human interference and produces content based on player preferences, it essentially lacks the means for clearing unwanted noise that ultimately results in serious errors in the final output.

“Mario AI” competitions begin in 2010 with the aim of introducing a robust framework and comprehensive platform for implementation and comparison among different PCG techniques. Shaker et al. [32] publish a report on this first entry in competitions. Ben Weber, Tomoyuki Shimizu and Tomonori Hashiyama are among the top contributors in this tournament. A well-designed framework of comparison metrics and consistent implementation are among the features that elicit significant achievements from these tournaments. Similarly, General Video Game AI (GVGAI) is another series of competitions in the field of AI in video games. The first event is held in 2014 and Perez-Liebana et al. [33, 34] provide a report on these

competitions. Using Video Game Description Language (VGDL) first introduced by Schaul [35, 36], this competition manages to bring the community of AI researcher together with inclusion of more categories like two-player games, learning and level generation. While VGDL is simple and easy to use, it has several shortcomings in phenotype-genotype mapping. The main problem with this descriptive framework is that it provides a high-level view of the problem and thus limiting the access to solution space. In addition, GVGAI incorporates a simplistic implementation, which is suitable for evaluating general AI agents, while Infinite Mario Bros. (IMB) used in Mario AI specializes in in-depth expressive range analysis in platformer games.

2.2.1 Probabilistic modeling

The idea of probabilistic modeling has been explored in the context of PCG for games in several studies including quality diversity and machine learning based approaches. While not necessarily in the scope of current study, these are essentially helpful in positioning the proposed method amongst the current body of work.

Quality Diversity (QD) algorithms are designed to provide a set of diverse solutions with defined quality from the behaviour space of the evolutionary population rather than focusing on finding a single global optimum in the latter space [37]. These approaches have been successfully applied to the domain of digital games with MAP-Elites (short for Multi-dimensional Archive of Phenotypic Elites—ME). Fontaine et al. [8] use the probabilistic modeling functionalities of Covariance Matrix Adaptation: Evolution Strategy (CMA-ES) to increase the performance of ME in searching the latent space of trained generative adversarial networks and extract interesting levels close to the work of a human. The proposed CMA-ME method efficiently performs maximal exploration of distorted behaviour space. However, in comparison in two domains, CMA-ES finds better individuals, but converges consistently.

Snodgrass et al. [38] use higher-order Markov chains to model two dimensional maps and learn statistical patterns contained in high quality human authored levels using different dependency matrices. In turn, the model is used to generate new content that exhibits the same statistical properties, with experimental validation and evaluation steps. The main challenge of this approach is in the amount of data needed to effectively estimate the probability distribution, which grows exponentially in higher-order Markov chain modeling. The main advantage of this approach is that it needs no hard-coded rules or constraints and operates solely on learned dependencies, generates diverse sets of maps, and shows good generalization in similar games. However, this method fails to replicate the overall structure of the maps, and lacks several vital DE. In the follow-up study, Snodgrass et al. [39] use three specific higher-order Markov models and turn to machine learning to exploit such models, while generalizing the resulting map generation to other games. This study highlights the performance of higher-order Markov chain models in building structural probabilistic models, which show fast sampling time favorable for online content generation, diverse sets of generated maps with mixed results that polarize the performance on two extremes of playable and aesthetically pleasing levels in

SMB game, compared to levels that are unplayable and unable to “capture long-range dependencies or global attributes” in the games *Kid Icarus* and *Loderunner*.

In a similar category as the above, Guzdial et al. [40] derive probabilistic graphical models from the gameplay videos that effectively capture the level design style of the original game. In this system, an unsupervised learning technique is used to learn the geometric sprite data from level chunks categorized from the gameplay videos as input. Using unsupervised learning to infer the accurate tile placements from human-authored original levels, along with the “level plan” component, contributes to this method’s effectiveness in capturing the semantic meaning of the level components.

From another perspective, Gonzalez-Duque et al. [41] use Bayesian optimization for the task of fast dynamic difficulty adjustment. Their system is a combination of two components, i.e. Gaussian process regression as a probabilistic regression model and an acquisition function. In detail, the system maintains a probabilistic regression model of the player, updates it based on the gameplay data and identifies the best level (using a modified acquisition function) in terms of difficulty that is suitable for the player’s skill level. This method needs only a few iterations in gathering player data dynamically and adjusting the difficulty, and thus works fast with low mean error at the expense of high variance, which in overall outperforms all the baseline methods.

As can be deduced from this extensive body of work, there is no comprehensive method for PCG. Each individual approach has its unique strengths and weaknesses, and almost all of them struggle to pinpoint a compatible approach to include PEM principles in their systems. One possible source of this challenge is the absence of effective modeling features in their integrated AI methodology. In addition, human interference limitations, feature selection bottlenecks and indirect mapping complications in controllability are the most reported shortcomings in the PCG domain, which accentuate the need for applying a more compatible approach to PEM that can be controlled effectively and efficiently while ruling out unwanted intrusions in the automatic process. Moreover, EDA variants and probabilistic modeling have proven useful in addressing these issues and their effects can be analyzed in an evolutionary process.

3 Proposed method

In this section, first we cover the main principles of EDA and provide a brief introduction of probabilistic modeling. Then, the proposed method and its complete description will be provided.

3.1 Estimation of distribution algorithm

The evolutionary method used in this study is based on EDA, which is a stochastic optimization method that is first introduced in 1994 by S. Baluja and includes iterative building and sampling of probabilistic models during an evolutionary optimization

process [42]. In contrast to other stochastic search-based optimization methods, EDA provides clear and explicit computational models of how to reach the optimum using its unique probabilistic models, which consequently provide a comprehensive overview of the problem domain. These models, in turn, can be used to control the evolutionary process and also, view and analyze the algorithm's performance. By logic, having extensive models for complex and multi-objective optimization problems at vast and discrete scales can be beneficial.

As laid out by Pelikan et al. [43], various EDA approaches similarly model the problem domain by computing the probability distribution for each variable, but they are evidently different in terms of creating probability dependencies among problem variables. The approaches that compute the probability of occurrence of each variable independently, provide no structural information about the problem setting, thus providing a parametric and single-dimensional representation. One good example for this class of EDAs is the Univariate Marginal Distribution Algorithm (UMDA), first introduced by H. Muhlenbein et al. in 1996 [2] which is capable of operating on discrete problem settings. Figure 1 shows the pseudo-code for basic UMDA. In this algorithm, parametric modeling enables calculating the distributions for each problem variable independently. Efficient flexibility in mapping between evolutionary algorithm and problem domain is considered another advantage of this method.

The setting for calculating probability distributions in UMDA is as follows. The selected promising candidate solutions are defined as a string of random variables (X_1, X_2, \dots, X_n) with length $n > 0$. In turn, the value for probability distribution $M(t)$ for simultaneous occurrence of each variable in string (X_1, X_2, \dots, X_n) can be formulated as shown in Eq. 1. This model provides explicit overview of the evolutionary population in each evolutionary generation.

$$\Pr(X_1, X_2, \dots, X_n) = \prod_{i=1}^n \Pr(X_i) \quad (1)$$

3.2 EDAPCG

In order to address some of the challenges in the field of PCG pointed out in Sect. 2, we propose a novel method called Estimation of Distribution Algorithm Procedural

Fig. 1 Pseudo-code for basic UMDA

1. $t \leftarrow 0$
2. generate population $P(0)$ of random solutions
3. while termination criterion is not satisfied, repeat
4. evaluate all candidate solutions in $P(t)$
5. select promising solutions $S(t)$ from $P(t)$
6. build a probabilistic model $M(t)$ for $S(t)$
7. generate new solutions $O(t)$ by sampling $M(t)$
8. create $P(t+1)$ by combining $O(t)$ and $P(t)$
9. $t \leftarrow t+1$

Content Generator (EDAPCG). EDAPCG incorporates the unique specifications of UMDA for probabilistic modeling with the aim of augmenting PCG in a more meaningful manner. The enjoining of generic PEM principles and succinct knowledge over the problem space provides unique possibilities to push the boundaries of this field. By introducing some level of automation to feed said knowledge into the system, EDAPCG offers an alternative to the need for feature selection, increasing the effectiveness of PEM. In addition, in EDAPCG, we perceive the problem of PCG as a combinatorial optimization problem that entails discrete optimization, which requires integer programming. This rationale enables direct mapping between genotype and phenotype to secure full access to the problem space as well as control over generated game content, resulting in better analysis of the expressive range of the proposed method.

The task of EDAPCG is to solve the discrete optimization problem and generate optimized game levels. Similar to the knapsack problem, there is a fixed length ($n=100$) string of discrete variables (DE), and EDAPCG has to provide a combination of DE in a way that provides valid solutions while optimizing the defined expressivity metrics using specific functionalities of UMDA. An overview of EDAPCG is shown in Fig. 2. As can be seen, EDAPCG is composed of three main parts, namely schematics, probabilistic estimation of distribution model and evaluation module, with each part addressing the challenges identified in the previous section. These parts will be described in the sections below, concluding with a step-by-step example.

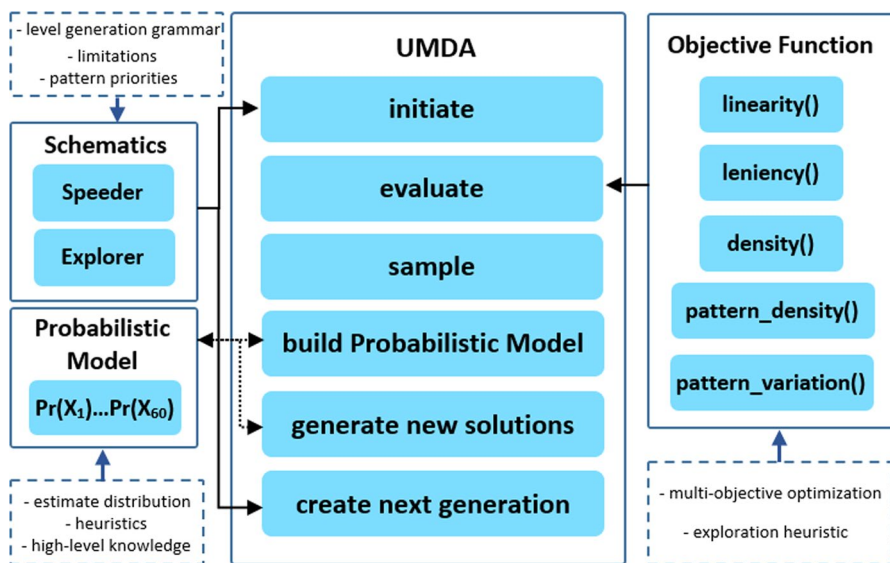


Fig. 2 An overview of the proposed system design – solid rectangles are the main and dashed counterparts are their respective functionalities in EDAPCG

3.2.1 Representation

The procedurally generated content in this study are playable game levels that run on the experimental framework IMB. Each level is a string of discrete DE with length of $n=100$, that will be considered as chromosomes in the evolutionary process. These DE include all the standard sprites used in the original SMB along with various combinations thereof (Appendix A). Figure 3 depicts the phenotype-genotype mapping and some examples on how various DE combinations form meso-patterns. Placement of repeated elements in the string is allowed, but the final output must be playable, otherwise it will be discarded as an impaired chromosome.

3.2.2 Generation

The initial population is generated randomly, but this random generation should be strictly according to schematics so that the initial content can be rendered as playable. Aside from having required grammar rules for content generation that increase the controllability of the output, the built-in PEM module of IMB is used in tandem with these schematics to prioritize the use of certain DE and patterns from the original game to contribute to generating levels for two distinct playstyles of “speeder” and “explorer.” In speeder mode, the emphasis is on creating obstacles, while in explorer mode enemy sprites and collectible DE are more prevalent. The schematics are coded manually by the authors according to the related study by S. Dahlskog et al. [30] for pattern creation. Each generated level string is formed from left to right, beginning with START and concluding with END at the length of $n=100$ (Fig. 3). Figure 4 lists some of the grammar rules used for generating levels (for a more comprehensive list, refer to S. Dahlskog et al. [30]).

3.2.3 Evaluation

After generating the individuals of the population, the objective function will be applied for evaluation. The objective function used in this method is a combination of evaluation (expressivity) metrics. The purpose of EDAPCG is to optimize the level generation process via a synthetic objective function (Fitness Function—FF) as laid out in Eq. 2.

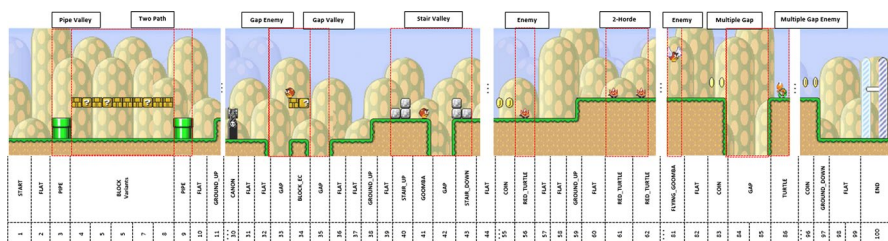


Fig. 3 Generated level sample (phenotype decoding), design elements (genotype encoding – dashed rectangle) and example meso-patterns (solid rectangle)

| Speeder Schematic | | | | | Explorer Schematic | | | | |
|-------------------|-----|--------------------------|--------|-------------------|--------------------|-----|--------------------------|--------|-------------------|
| pattern | wt. | grammar rule(s) | wt. | metric | pattern | wt. | grammar rule(s) | wt. | metric |
| - | | GROUND_UP GROUND_DOWN | 2 2 | linearity | - | | GROUND_UP GROUND_DOWN | 4 4 | linearity |
| enemy | 1 | i.e. GOOMBA | 2 | leniency | enemy | 3 | i.e. GOOMBA | 2 | leniency |
| | | 2-horde | 1 | density | | | 2-horde | 2 | density |
| | | 3-horde | 1 | pattern- | | | 3-horde | 2 | pattern- |
| | | 4-horde | 1 | variation/density | | | 4-horde | 2 | variation/density |
| gap | 3 | GAP | 2 | leniency | gap | 1 | GAP | 2 | leniency |
| | | multiple gaps | 3 | density | | | multiple gaps | 1 | density |
| | | (max 3 GAP) | | pattern- | | | (max 3 GAP) | | pattern- |
| | | gap enemy | 1 | variation/density | | | gap enemy | 2 | variation/density |
| | | pillar gap | 1 | | | | pillar gap | 1 | |
| valley | 2 | gap | 1 | | | 1 | gap | 1 | |
| | | valley | 2 | leniency | | | valley | 1 | leniency |
| | | pipe valley | 2 | density | | | pipe valley | 1 | density |
| | | enemy valley | 1 | pattern- | | | enemy valley | 2 | pattern- |
| | | roof valley | 1 | variation/density | | | roof valley | 1 | variation/density |
| multi-path | 1 | 2-path | 1 | leniency | multi-path | 2 | 2-path | 2 | leniency |
| | | 3-path | 1 | density | | | 3-path | 1 | density |
| | | risk and reward | 1 | pattern- | | | risk and reward | 2 | pattern- |
| | | | 1 | variation/density | | | | 2 | variation/density |
| stair | 2 | STAIRS_UP | 1 | density | stair | 1 | STAIRS_UP | 1 | density |
| | | STAIRS_DOWN | 1 | pattern- | | | STAIRS_DOWN | 1 | pattern- |
| pillar gap | 1 | | | variation/density | pillar gap | 1 | | | variation/density |
| | | gap | 1 | leniency | | | gap | 1 | leniency |
| | | PIPE | 1 | density | | | PIPE | 1 | density |
| | | gap | 1 | pattern- | | | gap | 1 | pattern- |
| stair valley | 1 | | | variation/density | stair valley | 1 | | | variation/density |
| | | enemy stair valley | 1 | leniency | | | enemy stair valley | 2 | leniency |
| | | empty stair valley | 1 | density | | | empty stair valley | 1 | density |
| | | gap stair valley | 2 | pattern- | | | gap stair valley | 1 | pattern- |
| | | | | variation/density | | | | | variation/density |

Fig. 4 Schematics' design for speeder and explorer playstyle, patterns (lowercase) are selected based on the empirically tuned weights and filled with various DE (uppercase) and/or pattern extension according to the laid out grammar rules – the affected metrics by each pattern are deduced logically

$$\mathbf{FF} = \alpha_1 \text{linearity} + \alpha_2 \text{Leniency} + \alpha_3 \text{Density} + \alpha_4 \text{PatternDensity} + \alpha_5 \text{PatternVariation}$$

$$0 < \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5 \leq 1$$

$$\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 = 1$$

(2)

The applied objective function is a maximization function and aims to maximize each of the components in Eq. 2, with the exception of the linearity metric, which is a minimization task and reversed to match the overall function. Coefficients α_1 , α_2 , α_3 , α_4 and α_5 determine the priority of the respective metrics. These coefficients can take values between 0 and 1. The total sum of these coefficients is 1, and the final fitness is normalized between 0 and 1 for all individuals. As can be seen in Eq. 2, the objective in this design is the combinatorial maximization of a weighted array of level metrics. The maximum values for each metric show the capacity of EDAPCG to reach that goal for tailoring levels for different playstyles. Considering that the current proposed method uses fixed and pre-defined playstyles for this task, setting them as targets for the generative system, optimizing the metrics to reach the target level structure would be a basic way of evaluating the system's output.

The expressivity metrics

The expressivity metrics used for comparison and classification of the previous methods are based on Horn et al. [3], except the density metric which is defined according to more recent Summerville et al. [44]. In this study, the density amount of the vertical level segments is not determined based on how many “platform” DE are therein, but is deduced via the updated definition on how many

varied DE are used in total [44]. Other metrics such as line critic introduced in Smith et al. [22], or the additional metrics proposed in Summerville et al. [44] will not be used, mainly because these metrics are more abstract and theoretical and lack the precise and practical reproduction details. The applied expressivity metrics along with descriptions of how they are calculated in this study are provided in Table 1. In addition, in order to measure the structural similarity in the expressive range of each generator, compression distance metric is used. This metric is calculated based on the description provided by Shaker et al. [7], in which levels are converted into a string of discrete values to identify different DE and changes in level linearity. Next, the diversity of generated output is measured using normalized compression distance (NCD) measure [7].

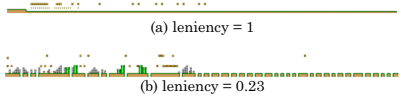
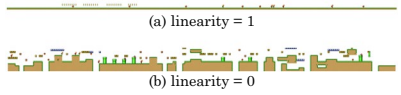
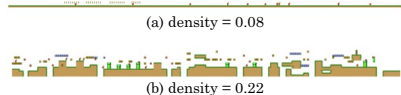
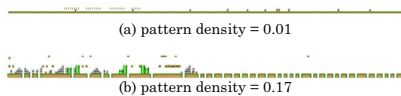
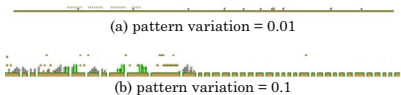
3.2.4 Selection

Picking promising individuals of each generation is based on their score obtained from the objective function in Eq. 2. In this step, selected individuals will be used for sampling and building the probabilistic model. At the end of each generation, 40 promising individuals are selected for the purpose of building models and transferring to the next generation.

3.2.5 Replacement

After transferring 40 promising individuals from the past generation based on their objective function score, 60 new individuals will be generated for the total sum of

Table 1 Expressivity metric description (all images taken from [3])

| | |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>Leniency metric deduces how difficult a level is for the player; calculated by counting the challenging DE and normalized for level length</p> |  <p>(a) leniency = 1 (b) leniency = 0.23</p> |
| <p>Linearity shows how much a level structure follows a straight line, finding the differences between the heights of the ground levels, normalized for level length</p> |  <p>(a) linearity = 1 (b) linearity = 0</p> |
| <p>Density shows how many distinct DE are in a predefined vertical segment of levels, assigning higher values to levels with more DE present in a vertical segment</p> |  <p>(a) density = 0.08 (b) density = 0.22</p> |
| <p>Pattern density counts the meso-patterns present in the level that are similar to the original game, normalized for level length</p> |  <p>(a) pattern density = 0.01 (b) pattern density = 0.17</p> |
| <p>Pattern variation measures how many distinct patterns are present in a level, assigning higher values to levels that include more varied meso-patterns</p> |  <p>(a) pattern variation = 0.01 (b) pattern variation = 0.1</p> |

100 chromosomes for the new generation to preserve genetic diversity. Objective function will be applied to 60 new individuals and the evolutionary population of 100 will be sorted again. This replacement approach is continued until the termination criterion is met.

3.2.6 Sampling

Building the probabilistic model is the most vital step in EDAPCG. The probabilistic model is sampled based on the information extracted from the promising candidates of each generation. To gain a better understanding of the sampling process, we refer to the previous UMDA assumptions in this section. Selected promising candidates are strings of random variables (X_1, X_2, \dots, X_n) with length of $n > 0$. In EDAPCG, the generated levels are equivalent of strings of random variables (i.e. DE). Therefore, according to the principles of UMDA, Eq. 1 can be rewritten as follows for calculating the probability of each DE at its respective index:

$$\Pr(DE_1, DE_2, \dots, DE_{60}) = \prod_{i=1}^{60} \Pr(DE_i) \quad (3)$$

The probability in Eq. 3 is calculated for each DE present in 40 elite individuals of the evolutionary generations. Algorithm 1 depicts the modeling approach and Fig. 5 shows the complete process for probabilistic modeling. In this process, the overall probabilistic model for each generation will be calculated and fed back to the system as information of desirable individuals. The referred modeling scheme in EDAPCG is parametric and provides a computational model over the characteristics of promising candidates, meaning the probability distribution of DE that rendered those individuals elite.

Algorithm 1: probabilistic modeling

```

1: for  $g \in G$  do
2:   for  $e \in E$  do
3:     for  $x \in X$  do
4:        $\Pr(x) \leftarrow$  probability of simultaneous occurrences of  $x$  in  $e$ 
5:        $M(e) \leftarrow \text{sample } \Pr(x)$ 
6:     end for
7:   end for
8:    $M(g) \leftarrow$  mean value for each index in all  $M(e)$  of  $g$ 
9: end for

```

Where G ($n=300$) is the set of evolutionary generations (each $g \in G$ corresponds to a single generation from the set), E ($n=40$) is the set of each generation's elite individuals (each $e \in E$ corresponds to a single elite individual of its respective generation) and X ($n=60$) is the set of distinct DE used in the experiments (each $x \in X$ corresponds to each discrete DE—appendix A). In Algorithm 1, the probability for simultaneous occurrence of each $x \in X$ is defined as $\Pr(x)$, inferred from Eq. 3. In

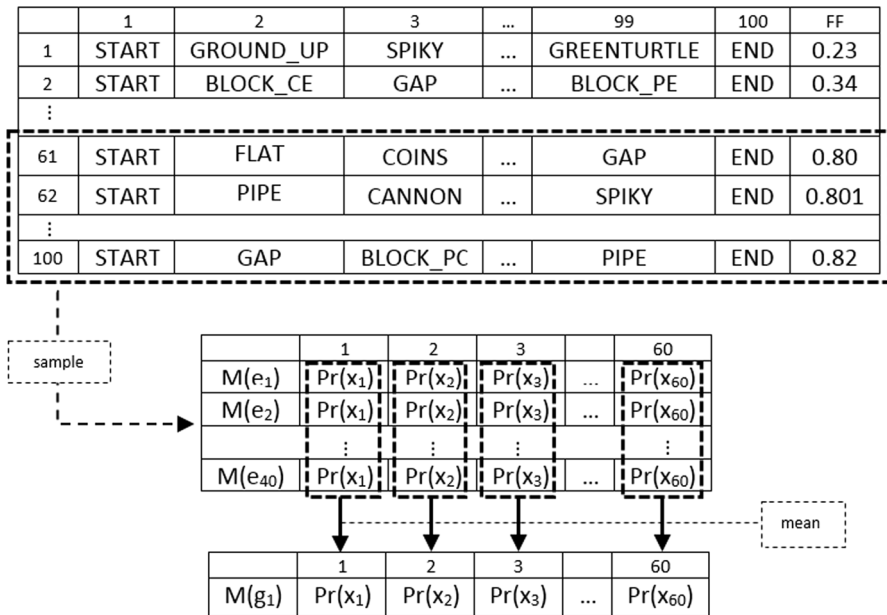


Fig. 5 Probabilistic modeling process for the first evolutionary generation (g_1). The population is sorted based on fitness function score (Eq. 2) and 40 best performing individuals are selected as the elite and sampled for probabilistic modeling— $M(e_1)$ refers to the probability distribution for the first elite individual and $M(g_1)$ refers to the resulting aggregate model for the first generation

turn, these individual probabilities are used to initialize/update the probability distribution model $M(e)$, which is the probability distribution array with length of $n = 60$, formed for $e \in E$. Similarly, $M(g)$ ($n = 60$) is the inferred aggregate probability distribution model of the elite population for each evolutionary generation, in which the mean values for $Pr(x)$ of each DE present in the elite is calculated and stored in their respective index (absent DE is presented as null). Figure 5 provides an overview of the probabilistic modeling scheme designed for EDAPCG.

3.2.7 Termination criterion

The termination criterion in EDAPCG is to reach 300 evolutionary generations, which is chosen empirically; in the majority of cases and configurations, the EDAPCG reaches an optimum within 300 generations. Next, the optimum in the final evolutionary population will be the output of EDAPCG, which will be mapped into a playable game level and provided on the IMB framework for the player to experience.

Example

The IMB framework includes a built-in PEM module that records the gameplay telemetry data using the built-in Notch level generator, which can be used to assign

one of the two playstyles (explorer/speeder) into the system as target player model. In order to build the new evolutionary generation, new solutions will be generated based on the respective schematics. Figure 4 depicts a non-comprehensive overview of the grammar rules for both schematics. Design patterns and individual DE have adjustable weights, which can be configured both manually and tweaked during the evolutionary process. In the next step, after generating the evolutionary population, the individuals will be evaluated based on the objective function (Eq. 2). However, in the current stage of experiments the coefficients for Eq. (2) are determined manually to tailor levels for each playstyle, a functionality that can be designed to be configured adaptively on the fly in later stages of the studies with human players.

Next, individuals in the evolutionary population will be sorted based on their objective function score and the 40 individuals with higher scores will be rendered as elite and sampled for parametric probabilistic modeling. Figure 5 depicts the parametric probabilistic modeling process of EDAPCG. The design logic behind this approach is to capture some of the behavioral characteristics of the elite solutions through the unique distribution of DE. In addition, a record of the past generations' elite can be archived and some parametric overview can be formed consisting of the mean and standard deviation values for individual probabilities of $x \in X$. This compact overview of probability distribution of various DE in the elite population across generations enables EDAPCG to (a) work in tandem with the objective function to guide the PCG process, (b) reduce similarities across generations and (c) provide an overview of the evolution of elite individuals to generate the final output.

At this stage, the population is evaluated for each expressivity metric and the overall performance of the elite individuals in using DE is recorded in $M(e)$. Next, inferring from the aggregated $M(g)$ and target metrics for the desired PCG preferences, weights for the next generation are tweaked, if needed. For example, if certain player preference demands speeder schematic for level generation, EDAPCG checks if the desired metrics for that are being satisfied, if not consults the $M(e)$ to see enough DE is used to replicate the target grammar rules, and tweaks the weights slightly in result. In more detail, presume that the speeder player type prefers less linearity. If the generation process is performing poorly in achieving the desired values for the respective metric, EDAPCG consults $M(e)$ for the probability distribution of vital DE to reach the target metric value (i.e. `GROUND_UP/GROUND_DOWN`). By multiplying the weight of the either individual DE (direct) or the respective grammar rules (indirectly), to a pre-defined value (i.e. 1.05), probability distribution will be biased for the next generation, resulting in effective exploration towards reaching target expressivity metric. This functionality is perceived as heuristics inferred from the probabilistic models and the values are determined empirically. It is worth mentioning that the distinction between two versions of probabilistic models is that $M(g)$ is used for gaining an overall overview of the past generations and generating the new evolutionary population, while $M(e)$ is used for more detailed inquiry into the behavioral characteristics of the elite individuals in each generation.

In turn, in order to avoid weakening the method's exploration functionality, these heuristics must be kept at negligible levels to ensure the evolutionary populations' diversity (uniform optimization of all the metrics) and prevent premature convergence of the solutions. This proved to be more important in the current problem

setting, as multi-objective optimization requires satisfying a complex trade-off between all the metrics, and hard bias in one metric results in blocking the evolutionary process from reaching more values in other metrics. It is worth mentioning that the definition of controllability used in this study is based on Horn et al. [3], according to which, the controllability module of EDAPCG is a hybrid of indirect and parameterized approaches. It is indirect, similar to the GE method as the coefficients of the evolutionary objective function can be adjusted, and it is parametrized because the weights of DE and each meso/macro pattern can be altered as well.

4 Experimental results

In this section, the experimental results obtained from the setting described in the previous section will be provided. In addition, several methodologies for in-depth and visual comparison are included.

Experimental testbed

The majority of studies in the field of PCG in platformer video games are implemented on the IMB testbed. Markus Persson authors this testbed for experimental and other recreational purposes. IMB became widely used after the adaptations of G. Smith and Sergey Karakovskiy for Mario A.I. competitions [32].

Data

The data, statistical information and expressive range principles used in the section are based on B. Horn et al. 2014 along with additional explanations provided in the previous section. The authors provide all the required information and framework for uniform comparison among various PCG methods based on the IMB framework [3].

4.1 Evaluation and comparison

In this sub-section, first the implementations of the proposed method will be evaluated. Next, the practical results of EDAPCG are provided and a comprehensive comparison with all the previously presented methods in this research field is performed. The various hardware and software specifications used in implementation are presented below.

| | |
|------------------|------------------------------------------------------------------|
| Operating system | Microsoft Windows ^(R) 8 / 64 bit (2013) |
| CPU | Intel ^(R) Core ^(TM) i5-5200U @ 2.20 GHz×64 |
| RAM | 6.00 GB |
| Java | Java ^(TM) SE Runtime [1.8.0 _201] |
| Ant | Apache Ant ^(TM) [1.10.5] (10 July 2018) |
| cmd | Microsoft Windows ^(R) [6.3.9600] (2013) |

Evaluation

In order to verify the performance of the proposed method, different tests have been done, i.e. reliability, convergence and stability. These evaluations are essential because at its core, EDAPCG is a stochastic optimization method. In reliability, different sets of coefficients are applied to the objective function in order to verify its robust performance, which can be deduced from the comparisons later in this section. In addition, convergence reflects the upward trend of the overall objective function scores for individuals in each generation towards reaching the elites' scores. To clarify, the generation number of 49 for Fig. 6 is selected based on empirical trial and observation and for summarization, as the statistics hold unchanged after this number of generations. In Fig. 6, statistics for the elite population (green) and mean values of each evolutionary generation's objective function score (blue) depict the convergence pattern for EDAPCG. Moreover, EDAPCG's stability in producing results can be deduced from Fig. 7, in which negligible standard deviation values of objective function score of evolutionary generations are shown in 5 consecutive runs.

Comparison

Table 2 shows the overall results of the proposed method. The count of evolutionary generations is set to 300 (termination criterion). In the first generation, the objective function calculations are done for every 100 individuals. In the next generations, these calculations are done for the 60 new individuals, which adds up to 18,040 total objective function calculations. Coefficients for the objective function is constant across each run and are as follow: $\alpha_1=0.4$, $\alpha_2=0.2$, $\alpha_3=0.2$, $\alpha_4=0.1$, $\alpha_5=0.1$, which deduced based on trial and error process to find the optimal settings for the approach. The compression distance metric is calculated based on all of the generated levels in the evolutionary process, because of this fundamental difference in calculation, it is not placed along other metrics.

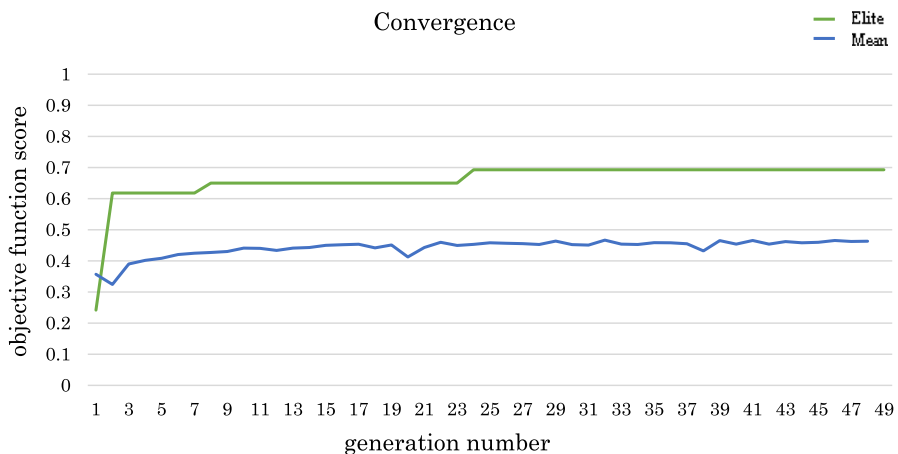


Fig. 6 Convergence diagram

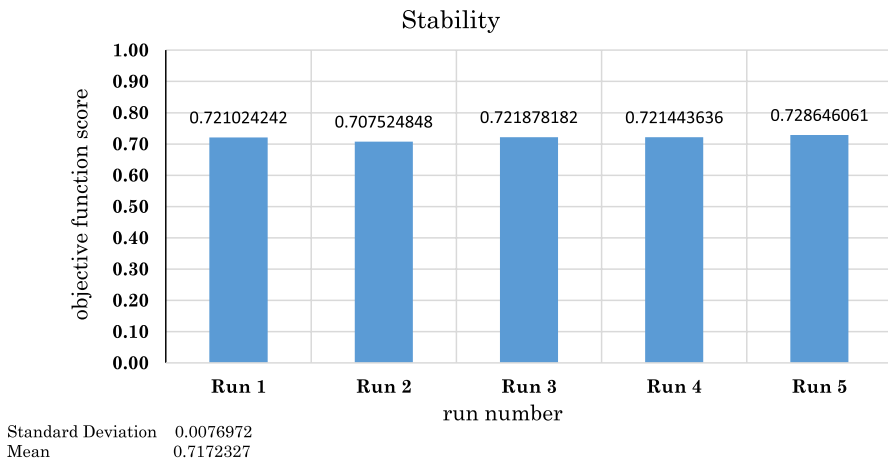


Fig. 7 Stability diagram

Table 2 EDAPCG results

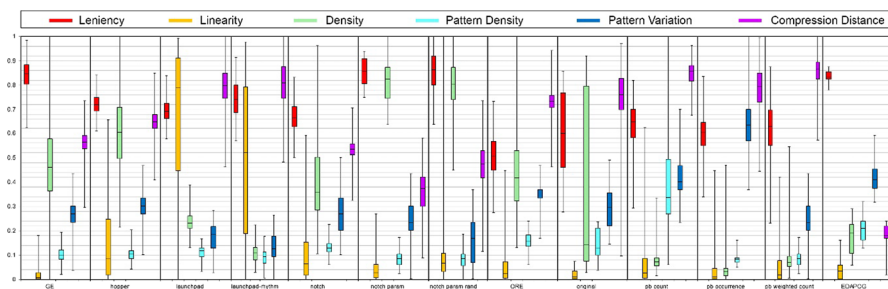
| # | Leniency | Linearity | Density | Pattern density | Pattern variation |
|--------------------|------------|-------------|------------|-----------------|-------------------|
| 1 | 0.8701 | 0 | 0.08 | 0.13 | 0.318181818 |
| 2 | 0.8183 | 0 | 0.18 | 0.17 | 0.318181818 |
| 3 | 0.8302 | 0 | 0.1 | 0.16 | 0.409090909 |
| 4 | 0.8379 | 0.006666667 | 0.06 | 0.16 | 0.409090909 |
| 5 | 0.8752 | 0.161333333 | 0.13 | 0.24 | 0.454545455 |
| 6 | 0.8577 | 0.061333333 | 0.29 | 0.31 | 0.590909091 |
| 7 | 0.7802 | 0.030666667 | 0.2 | 0.24 | 0.363636364 |
| 8 | 0.8454 | 0.053333333 | 0.24 | 0.23 | 0.409090909 |
| 9 | 0.8227 | 0.041333333 | 0.23 | 0.32 | 0.454545455 |
| 10 | 0.8308 | 0.065333333 | 0.22 | 0.19 | 0.545454545 |
| mean | 0.83685 | 0.042 | 0.173 | 0.215 | 0.427272727 |
| standard deviation | 0.02772801 | 0.04939236 | 0.07674634 | 0.06450667 | 0.08886593 |

In Table 3, results of the proposed method are included together with previous methods as provided in Horn et al. [3]. In each metric's column, the leftmost number is the mean value calculated from various runs and the rightmost number is the standard deviation. Figure 8 depicts the same statistic in a graphical manner, in the continuation of the figure included in Horn et al. [3] reproduced using provided data.

In order to establish a comparison point, each generator is introduced based on descriptions provided by Horn et al. [3]. *Notch* generates levels from left to right, based on predefined probabilities and basic checks to produce playable levels. Similarly, *Parameterized Notch* takes parameters for certain DE in order to bias how levels are generated. *Hopper* is also similar to Notch variants, but with adaptability in which level generation can be altered based on the players' prior performance.

Table 3 Statistical overview of EDAPCG and previous methods

| Generator | Leniency | | Linearity | | Density | | Pattern density | | Pattern variation | | Pattern variation | |
|-------------------|-------------|--------|-------------|--------|-------------|--------|-----------------|--------|-------------------|--------|-------------------|--------|
| GE | 0.84 | (0.06) | 0.02 | (0.03) | 0.47 | (0.16) | 0.1 | (0.03) | 0.27 | (0.06) | 0.56 | (0.04) |
| hopper | 0.72 | (0.04) | 0.15 | (0.16) | 0.6 | (0.15) | 0.1 | (0.02) | 0.29 | (0.05) | 0.65 | (0.05) |
| Launchpad | 0.7 | (0.05) | 0.66 | (0.31) | 0.24 | (0.04) | 0.11 | (0.03) | 0.17 | (0.05) | 0.8 | (0.07) |
| Launchpad-rhythm | 0.74 | (0.07) | 0.49 | (0.32) | 0.11 | (0.04) | 0.09 | (0.03) | 0.13 | (0.06) | 0.81 | (0.09) |
| notch | 0.67 | (0.06) | 0.1 | (0.11) | 0.4 | (0.16) | 0.13 | (0.02) | 0.27 | (0.08) | 0.53 | (0.03) |
| notch param | 0.85 | (0.06) | 0.04 | (0.05) | 0.81 | (0.08) | 0.08 | (0.03) | 0.24 | (0.07) | 0.36 | (0.08) |
| notch param rand | 0.86 | (0.08) | 0.08 | (0.06) | 0.8 | (0.1) | 0.08 | (0.03) | 0.17 | (0.09) | 0.47 | (0.08) |
| ORE | 0.51 | (0.08) | 0.05 | (0.06) | 0.43 | (0.15) | 0.16 | (0.03) | 0.35 | (0.05) | 0.73 | (0.04) |
| original | 0.61 | (0.18) | 0.02 | (0.02) | 0.35 | (0.37) | 0.14 | (0.06) | 0.3 | (0.1) | 0.76 | (0.11) |
| pb count | 0.63 | (0.1) | 0.07 | (0.09) | 0.08 | (0.05) | 0.39 | (0.17) | 0.41 | (0.07) | 0.85 | (0.04) |
| pb occurrence | 0.6 | (0.08) | 0.04 | (0.06) | 0.06 | (0.09) | 0.08 | (0.02) | 0.64 | (0.11) | 0.79 | (0.08) |
| pb weighted count | 0.61 | (0.12) | 0.06 | (0.08) | 0.09 | (0.08) | 0.08 | (0.03) | 0.24 | (0.07) | 0.86 | (0.05) |
| EDAPCG | 0.83 | (0.02) | 0.03 | (0.03) | 0.17 | (0.07) | 0.21 | (0.06) | 0.42 | (0.08) | 0.82 | (0.03) |

**Fig. 8** Graphical representation of previous methods and EDAPCG with proposed coefficients

Launchpad and its rhythm-based variant are level generators that use design grammars inferred from the original levels with hard-coded constraints to generate valid output. *Occupancy-Regulated Extension* (ORE) pieces together small, hand-authored chunks of levels to generate playable levels. Pattern-based methods use evolutionary computation in generating sequences of micro-patterns inferred from the original levels with three distinct fitness functions for each variant, each aimed at increasing meso-pattern replication. In more detail, *pattern occurrence* counts each meso-pattern once, while *pattern count* is agnostic to repetition. In a slightly different approach, *weighted pattern count* prioritizes each pattern based on their rarity. *Grammatical Evolution* (GE) uses evolutionary computation to evolve and expand levels based on instructions inferred from design grammars, while evaluating the generated levels using a fitness function [3].

Table 4 Leniency

| Proposed coefficients | | | Standard coefficients | | | Uniform coefficients | | |
|-----------------------|----------|--------|-----------------------|----------|--------|----------------------|----------|--------|
| Generator | Leniency | | Generator | Leniency | | Generator | Leniency | |
| notch param rand | 0.86 | (0.08) | notch param rand | 0.86 | (0.08) | notch param rand | 0.86 | (0.08) |
| notch param | 0.85 | (0.06) | notch param | 0.85 | (0.06) | notch param | 0.85 | (0.06) |
| GE | 0.84 | (0.06) | GE | 0.84 | (0.06) | GE | 0.84 | (0.06) |
| EDAPCG | 0.83 | (0.02) | EDAPCG | 0.78 | (0.02) | EDAPCG | 0.79 | (0.04) |
| Launchpad-rhythm | 0.74 | (0.07) | Launchpad-rhythm | 0.74 | (0.07) | Launchpad-rhythm | 0.74 | (0.07) |
| hopper | 0.72 | (0.04) | hopper | 0.72 | (0.04) | hopper | 0.72 | (0.04) |
| Launchpad | 0.7 | (0.05) | Launchpad | 0.7 | (0.05) | Launchpad | 0.7 | (0.05) |
| notch | 0.67 | (0.06) | notch | 0.67 | (0.06) | notch | 0.67 | (0.06) |
| pb count | 0.63 | (0.1) | pb count | 0.63 | (0.1) | pb count | 0.63 | (0.1) |
| pb weighted count | 0.61 | (0.12) | pb weighted count | 0.61 | (0.12) | pb weighted count | 0.61 | (0.12) |
| original | 0.61 | (0.18) | original | 0.61 | (0.18) | original | 0.61 | (0.18) |
| pb occurrence | 0.6 | (0.08) | pb occurrence | 0.6 | (0.08) | pb occurrence | 0.6 | (0.08) |
| ORE | 0.51 | (0.08) | ORE | 0.51 | (0.08) | ORE | 0.51 | (0.08) |

Table 5 Linearity

| Proposed coefficients | | | Standard coefficients | | | Uniform coefficients | | |
|-----------------------|-----------|--------|-----------------------|-----------|--------|----------------------|-----------|--------|
| Generator | Linearity | | Generator | Linearity | | Generator | Linearity | |
| original | 0.02 | (0.02) | original | 0.02 | (0.02) | original | 0.02 | (0.02) |
| GE | 0.02 | (0.03) | GE | 0.02 | (0.03) | GE | 0.02 | (0.03) |
| EDAPCG | 0.03 | (0.03) | notch param | 0.04 | (0.05) | notch param | 0.04 | (0.05) |
| notch param | 0.04 | (0.05) | pb occurrence | 0.04 | (0.06) | pb occurrence | 0.04 | (0.06) |
| pb occurrence | 0.04 | (0.06) | ORE | 0.05 | (0.06) | ORE | 0.05 | (0.06) |
| ORE | 0.05 | (0.06) | pb weighted count | 0.06 | (0.08) | pb weighted count | 0.06 | (0.08) |
| pb weighted count | 0.06 | (0.08) | pb count | 0.07 | (0.09) | pb count | 0.07 | (0.09) |
| pb count | 0.07 | (0.09) | notch param rand | 0.08 | (0.06) | notch param rand | 0.08 | (0.06) |
| notch param rand | 0.08 | (0.06) | notch | 0.1 | (0.11) | notch | 0.1 | (0.11) |
| notch | 0.1 | (0.11) | hopper | 0.15 | (0.16) | hopper | 0.15 | (0.16) |
| hopper | 0.15 | (0.16) | EDAPCG | 0.19 | (0.05) | EDAPCG | 0.17 | (0.08) |
| Launchpad-rhythm | 0.49 | (0.32) | Launchpad-rhythm | 0.49 | (0.32) | Launchpad-rhythm | 0.49 | (0.32) |
| Launchpad | 0.66 | (0.31) | Launchpad | 0.66 | (0.31) | Launchpad | 0.66 | (0.31) |

Separate comparison of metrics

The ranked results of EDAPCG expressivity metrics compared to previous methods can be seen in Table 4 for leniency, Table 5 for linearity, Table 6 for density, Table 7 for pattern density, Table 8 for pattern variation and Table 9 for compression distance. For each metric, a separate set of coefficients for the

Table 6 Density

| Proposed coefficients | | | Standard coefficients | | | Uniform coefficients | | |
|-----------------------|---------|--------|-----------------------|---------|--------|----------------------|---------|--------|
| Generator | Density | | Generator | Density | | Generator | Density | |
| notch param | 0.81 | (0.08) | notch param | 0.81 | (0.08) | notch param | 0.81 | (0.08) |
| notch param rand | 0.8 | (0.1) | notch param rand | 0.8 | (0.1) | notch param rand | 0.8 | (0.1) |
| hopper | 0.6 | (0.15) | hopper | 0.6 | (0.15) | hopper | 0.6 | (0.15) |
| GE | 0.47 | (0.16) | GE | 0.47 | (0.16) | GE | 0.47 | (0.16) |
| ORE | 0.43 | (0.15) | ORE | 0.43 | (0.15) | ORE | 0.43 | (0.15) |
| original | 0.35 | (0.37) | original | 0.35 | (0.37) | original | 0.35 | (0.37) |
| Launchpad | 0.24 | (0.04) | Launchpad | 0.24 | (0.04) | EDAPCG | 0.25 | (0.08) |
| EDAPCG | 0.17 | (0.07) | EDAPCG | 0.22 | (0.06) | Launchpad | 0.24 | (0.04) |
| Launchpad-rhythm | 0.11 | (0.04) | Launchpad-rhythm | 0.11 | (0.04) | Launchpad-rhythm | 0.11 | (0.04) |
| pb weighted count | 0.09 | (0.08) | pb weighted count | 0.09 | (0.08) | pb weighted count | 0.09 | (0.08) |
| pb count | 0.08 | (0.05) | pb count | 0.08 | (0.05) | pb count | 0.08 | (0.05) |
| pb occurrence | 0.06 | (0.09) | pb occurrence | 0.06 | (0.09) | pb occurrence | 0.06 | (0.09) |
| notch | 0.4 | (0.16) | notch | 0.4 | (0.16) | notch | 0.4 | (0.16) |

Table 7 Pattern density

| Proposed coefficients | | | Standard coefficients | | | Uniform coefficients | | |
|-----------------------|-----------------|--------|-----------------------|-----------------|--------|----------------------|-----------------|--------|
| Generator | Pattern density | | Generator | Pattern density | | Generator | Pattern density | |
| pb count | 0.39 | (0.17) | pb count | 0.39 | (0.17) | pb count | 0.39 | (0.17) |
| EDAPCG | 0.21 | (0.06) | EDAPCG | 0.33 | (0.02) | EDAPCG | 0.3 | (0.07) |
| ORE | 0.16 | (0.03) | ORE | 0.16 | (0.03) | ORE | 0.16 | (0.03) |
| original | 0.14 | (0.06) | original | 0.14 | (0.06) | original | 0.14 | (0.06) |
| notch | 0.13 | (0.02) | notch | 0.13 | (0.02) | Notch | 0.13 | (0.02) |
| Launchpad | 0.11 | (0.03) | Launchpad | 0.11 | (0.03) | Launchpad | 0.11 | (0.03) |
| hopper | 0.1 | (0.02) | hopper | 0.1 | (0.02) | Hopper | 0.1 | (0.02) |
| GE | 0.1 | (0.03) | GE | 0.1 | (0.03) | GE | 0.1 | (0.03) |
| pb occurrence | 0.08 | (0.02) | pb occurrence | 0.08 | (0.02) | pb occurrence | 0.08 | (0.02) |
| pb weighted count | 0.08 | (0.03) | pb weighted count | 0.08 | (0.03) | pb weighted count | 0.08 | (0.03) |
| notch param rand | 0.08 | (0.03) | notch param rand | 0.08 | (0.03) | notch param rand | 0.08 | (0.03) |
| notch param | 0.08 | (0.03) | notch param | 0.08 | (0.03) | notch param | 0.08 | (0.03) |
| Launchpad-rhythm | 0.09 | (0.03) | Launchpad-rhythm | 0.09 | (0.03) | Launchpad-rhythm | 0.09 | (0.03) |

Table 8 Pattern variation

| Proposed coefficients | | | Standard coefficients | | | Uniform coefficients | | |
|-----------------------|-------------------|--------|-----------------------|-------------------|--------|----------------------|-------------------|--------|
| Generator | Pattern variation | | Generator | Pattern variation | | Generator | Pattern variation | |
| pb occurrence | 0.64 | (0.11) | pb occurrence | 0.64 | (0.11) | pb occurrence | 0.64 | (0.11) |
| EDAPCG | 0.43 | (0.09) | EDAPCG | 0.56 | (0.04) | EDAPCG | 0.53 | (0.11) |
| pb count | 0.41 | (0.07) | pb count | 0.41 | (0.07) | pb count | 0.41 | (0.07) |
| ORE | 0.35 | (0.05) | ORE | 0.35 | (0.05) | ORE | 0.35 | (0.05) |
| original | 0.3 | (0.1) | original | 0.3 | (0.1) | original | 0.3 | (0.1) |
| hopper | 0.29 | (0.05) | hopper | 0.29 | (0.05) | hopper | 0.29 | (0.05) |
| GE | 0.27 | (0.06) | GE | 0.27 | (0.06) | GE | 0.27 | (0.06) |
| notch | 0.27 | (0.08) | notch | 0.27 | (0.08) | notch | 0.27 | (0.08) |
| notch param | 0.24 | (0.07) | notch param | 0.24 | (0.07) | notch param | 0.24 | (0.07) |
| pb weighted count | 0.24 | (0.07) | pb weighted count | 0.24 | (0.07) | pb weighted count | 0.24 | (0.07) |
| Launchpad | 0.17 | (0.05) | Launchpad | 0.17 | (0.05) | Launchpad | 0.17 | (0.05) |
| notch param rand | 0.17 | (0.09) | notch param rand | 0.17 | (0.09) | notch param rand | 0.17 | (0.09) |
| Launchpad-rhythm | 0.13 | (0.06) | Launchpad-rhythm | 0.13 | (0.06) | Launchpad-rhythm | 0.13 | (0.06) |

Table 9 Compression distance

| Proposed coefficients | | | Standard coefficients | | | Uniform coefficients | | |
|-----------------------|----------------------|---------------|-----------------------|----------------------|---------------|----------------------|----------------------|---------------|
| Generator | Compression distance | | Generator | Compression distance | | Generator | Compression distance | |
| pb weighted count | 0.86 | (0.05) | pb weighted count | 0.86 | (0.05) | pb weighted count | 0.86 | (0.05) |
| pb count | 0.85 | (0.04) | EDAPCG | 0.85 | (0.04) | pb count | 0.85 | (0.04) |
| EDAPCG | 0.82 | (0.03) | pb count | 0.85 | (0.04) | EDAPCG | 0.83 | (0.03) |
| Launchpad-rhythm | 0.81 | (0.09) | Launchpad-rhythm | 0.81 | (0.09) | Launchpad-rhythm | 0.81 | (0.09) |
| Launchpad | 0.8 | (0.07) | Launchpad | 0.8 | (0.07) | Launchpad | 0.8 | (0.07) |
| pb occurrence | 0.79 | (0.08) | pb occurrence | 0.79 | (0.08) | pb occurrence | 0.79 | (0.08) |
| original | 0.76 | (0.11) | original | 0.76 | (0.11) | original | 0.76 | (0.11) |
| ORE | 0.73 | (0.04) | ORE | 0.73 | (0.04) | ORE | 0.73 | (0.04) |
| hopper | 0.65 | (0.05) | hopper | 0.65 | (0.05) | hopper | 0.65 | (0.05) |
| GE | 0.56 | (0.04) | GE | 0.56 | (0.04) | GE | 0.56 | (0.04) |
| notch | 0.53 | (0.03) | notch | 0.53 | (0.03) | notch | 0.53 | (0.03) |
| notch param rand | 0.47 | (0.08) | notch param rand | 0.47 | (0.08) | notch param rand | 0.47 | (0.08) |
| notch param | 0.36 | (0.08) | notch param | 0.36 | (0.08) | notch param | 0.36 | (0.08) |

objective function is used to test the performance of the proposed method in different settings. The standard coefficients are based upon the priorities set in the related papers.

| | |
|-----------------------|----------------------------------------------------------------------------------|
| Uniform coefficients | $\alpha_1 = 0.2, \alpha_2 = 0.2, \alpha_3 = 0.2, \alpha_4 = 0.2, \alpha_5 = 0.2$ |
| Standard coefficients | $\alpha_1 = 0.1, \alpha_2 = 0.2, \alpha_3 = 0.3, \alpha_4 = 0.2, \alpha_5 = 0.2$ |
| Proposed coefficients | $\alpha_1 = 0.4, \alpha_2 = 0.2, \alpha_3 = 0.2, \alpha_4 = 0.1, \alpha_5 = 0.1$ |

Table 4 shows the performance of all methods in the leniency metric. Both notch and GE show high scores in this metric, while pattern-based methods replicate a wide leniency range of the original game. Another important factor to have in mind regarding leniency is the ability of content generator to control the difficulty level and tailor it to the needs of each individual player. In this regard, EDAPCG shows a limited range of different leniency values. Overall, EDAPCG produces lenient levels; however, they are according to the heuristics set in the schematics and contribute to the proposed method's controllability, which is not fixed and can be altered. For more experiments on this, refer to the *sample levels* sub-section.

The best performance in linearity metric belongs to the levels of the original game, mainly because of highly creative designing by humans. Next is GE, which uses platforms and various DE unlimitedly, resulting in non-linear levels. EDAPCG can be competitive in this metric, given it uses carefully configured coefficients in its objective function. However, in other configurations, the proposed method produces linear levels that rank just above Launchpad at the end of the linearity rankings. An overview for the performance of all methods can be seen in Table 5.

As can be seen in Table 6, EDAPCG's modest performance in density metric is in part due to inherent limitations in its implementation. As can be seen in Fig. 3 the number of different DE in each segment is limited compared to other methods. Constructive methods like notch, hopper and ORE use DE without any limitations and add more on each iteration, thus performing well in this metric. Similarly, GE and Launchpad do not have any limitations on DE usage; therefore perform well in this regard. On the other hand, content generators that use DE particularly with the aim of forming meso-patterns are susceptible to perform less desirably in density metric, mainly because they are designed to replicate as many patterns as possible, thus using each DE more restrictively. The proposed method can be classified in this category as it uses schematics and strict grammar rules to replicate the meso-patterns of the original game.

EDAPCG's capability in producing meso-patterns is apparent in Tables 7 and 8. While pattern-based count and pattern-based occurrence are two specialized methods in producing high number and unique meso-patterns, respectively, they tend to perform poorly in other expressivity metrics. EDAPCG's position alongside these methods is noteworthy considering its admissible rankings in other metric categories. Surprisingly, ORE shows great potential in replicating original game patterns, considering the fact that it is a constructive method. Moreover, the poor performances of Launchpad, hopper and notch are in part due to the fact that pattern-focused metrics are introduced after the introduction of these methods.

Table 9 shows the performance of the content generators in the compression distance metric. In the category of level distance metrics, compression distance measures the difference between two pairs of levels in the content generator's output [3]. EDAPCG's ability to perform well in this metric is twofold; first, it is an evolutionary method and can be configured to have great variation in solution generation. Second, the probabilistic modeling provides extensive knowledge about the estimation of distribution of DE in promising individuals of previous generations, which in turn informs the PCG systems and biases the generation function to produce different solutions for the next generation. This can be observed by visual inspection of sample levels in the next section. Interestingly, EDAPCG is placed on top and in close competition with pattern-oriented and rhythm-based methods, which points out the behaviour that the overall pattern count and variation may be linked to producing a variety of levels. In addition, the proposed method is ranked above the other evolutionary methods GE and ORE, which further reinforces the previous hypothesis.

Comparison using all metrics

As can be deduced from Tables 4, 5, 6, 7, 8 and 9, the majority of the previous methods perform well in some expressivity metrics, while showing moderate results in others. In contrast, EDAPCG shows promising results in most of the metrics. Moreover, it can be seen that an overall improvement is omnipresent in achieving all of the expressivity metrics. In order to inquire more on the performance of EDAPCG, results from the previous methods are inserted in the objective function proposed in EDAPCG (Eq. 2); these additional calculations provide a better overview on the

Table 10 Overall comparison based on proposed coefficients in objective function

| # | Generator | Leniency | Linearity | Density | Pattern density | Pattern variation | Compression distance | Score |
|----|-------------------|----------|-----------|---------|-----------------|-------------------|----------------------|--------------|
| 1 | EDAPCG | 0.83 | 0.04 | 0.17 | 0.21 | 0.42 | 0.82 | 1.605 |
| 2 | notch param | 0.85 | 0.04 | 0.81 | 0.08 | 0.24 | 0.36 | 1.406 |
| 3 | notch param rand | 0.86 | 0.08 | 0.8 | 0.08 | 0.17 | 0.47 | 1.272 |
| 4 | GE | 0.84 | 0.02 | 0.47 | 0.1 | 0.27 | 0.56 | 1.215 |
| 5 | notch | 0.67 | 0.1 | 0.4 | 0.13 | 0.27 | 0.53 | 1.158 |
| 6 | hopper | 0.72 | 0.15 | 0.6 | 0.1 | 0.29 | 0.65 | 1.028 |
| 7 | original | 0.61 | 0.02 | 0.35 | 0.14 | 0.3 | 0.76 | 0.954 |
| 8 | ORE | 0.51 | 0.05 | 0.43 | 0.16 | 0.35 | 0.73 | 0.948 |
| 9 | pb occurrence | 0.6 | 0.04 | 0.06 | 0.08 | 0.64 | 0.79 | 0.936 |
| 10 | pb count | 0.63 | 0.07 | 0.08 | 0.39 | 0.41 | 0.85 | 0.878 |
| 11 | pb weighted count | 0.61 | 0.06 | 0.09 | 0.08 | 0.24 | 0.86 | 0.819 |
| 12 | Launchpad-rhythm | 0.74 | 0.49 | 0.11 | 0.09 | 0.13 | 0.81 | 0.682 |
| 13 | Launchpad | 0.7 | 0.66 | 0.24 | 0.11 | 0.17 | 0.8 | 0.603 |

The coefficients are $\alpha_1=0.4$, $\alpha_2=0.2$, $\alpha_3=0.2$, $\alpha_4=0.1$, $\alpha_5=0.1$

Table 11 Overall comparison based on uniform coefficients in objective function

| # | Generator | Leniency | Linearity | Density | Pattern density | Pattern variation | Compression distance | Score |
|----|-------------------|----------|-----------|---------|-----------------|-------------------|----------------------|-------------|
| 1 | EDAPCG | 0.79 | 0.17 | 0.25 | 0.3 | 0.53 | 0.83 | 1.37 |
| 2 | notch param | 0.85 | 0.04 | 0.81 | 0.08 | 0.24 | 0.36 | 1.228 |
| 3 | notch param rand | 0.86 | 0.08 | 0.8 | 0.08 | 0.17 | 0.47 | 1.096 |
| 4 | GE | 0.84 | 0.02 | 0.47 | 0.1 | 0.27 | 0.56 | 0.972 |
| 5 | notch | 0.67 | 0.1 | 0.4 | 0.13 | 0.27 | 0.53 | 0.944 |
| 6 | hopper | 0.72 | 0.15 | 0.6 | 0.1 | 0.29 | 0.65 | 0.862 |
| 7 | ORE | 0.51 | 0.05 | 0.43 | 0.16 | 0.35 | 0.73 | 0.75 |
| 8 | original | 0.61 | 0.02 | 0.35 | 0.14 | 0.3 | 0.76 | 0.716 |
| 9 | pb occurrence | 0.6 | 0.04 | 0.06 | 0.08 | 0.64 | 0.79 | 0.678 |
| 10 | pb count | 0.63 | 0.07 | 0.08 | 0.39 | 0.41 | 0.85 | 0.638 |
| 11 | pb weighted count | 0.61 | 0.06 | 0.09 | 0.08 | 0.24 | 0.86 | 0.532 |
| 12 | Launchpad | 0.7 | 0.66 | 0.24 | 0.11 | 0.17 | 0.8 | 0.512 |
| 13 | Launchpad-rhythm | 0.74 | 0.49 | 0.11 | 0.09 | 0.13 | 0.81 | 0.506 |

The coefficients are $\alpha_1=0.2$, $\alpha_2=0.2$, $\alpha_3=0.2$, $\alpha_4=0.2$, $\alpha_5=0.2$

Table 12 Overall comparison based on standard coefficients in objective function

| # | Generator | Leniency | Linearity | Density | Pattern density | Pattern variation | Compression distance | Score |
|----|-------------------|----------|-----------|---------|-----------------|-------------------|----------------------|--------------|
| 1 | EDAPCG | 0.78 | 0.19 | 0.22 | 0.33 | 0.56 | 0.85 | 1.342 |
| 2 | notch param | 0.85 | 0.04 | 0.81 | 0.08 | 0.24 | 0.36 | 1.14 |
| 3 | notch param rand | 0.86 | 0.08 | 0.8 | 0.08 | 0.17 | 0.47 | 1.012 |
| 4 | GE | 0.84 | 0.02 | 0.47 | 0.1 | 0.27 | 0.56 | 0.884 |
| 5 | notch | 0.67 | 0.1 | 0.4 | 0.13 | 0.27 | 0.53 | 0.867 |
| 6 | hopper | 0.72 | 0.15 | 0.6 | 0.1 | 0.29 | 0.65 | 0.787 |
| 7 | ORE | 0.51 | 0.05 | 0.43 | 0.16 | 0.35 | 0.73 | 0.671 |
| 8 | original | 0.61 | 0.02 | 0.35 | 0.14 | 0.3 | 0.76 | 0.632 |
| 9 | pb occurrence | 0.6 | 0.04 | 0.06 | 0.08 | 0.64 | 0.79 | 0.59 |
| 10 | pb count | 0.63 | 0.07 | 0.08 | 0.39 | 0.41 | 0.85 | 0.584 |
| 11 | Launchpad | 0.7 | 0.66 | 0.24 | 0.11 | 0.17 | 0.8 | 0.489 |
| 12 | Launchpad-rhythm | 0.74 | 0.49 | 0.11 | 0.09 | 0.13 | 0.81 | 0.464 |
| 13 | pb weighted count | 0.61 | 0.06 | 0.09 | 0.08 | 0.24 | 0.86 | 0.446 |

The coefficients are $\alpha_1=0.1$, $\alpha_2=0.2$, $\alpha_3=0.3$, $\alpha_4=0.2$, $\alpha_5=0.2$

findings of this paper. The resulting rankings are presented in Tables 10, 11 and 12. The EDAPCG values are transferred from Table 3 and the values for other methods are directly inserted into the Eq. (2) from the reference source [3].

Interestingly, the first position of EDAPCG in these comparisons is followed by notch variations and GE in all three tables. Notch as the default built-in PCG method in the IMB framework is the only approach that incorporates basic probabilistic principles in its content generation, and GE is the combination of grammar based PCG with evolutionary implementation. In conclusion, the fact that EDAPCG combines these two orientations and outperforms other methods in comparison, contributes to the acceptable results in achieving design objectives laid out early in Sect. 3. Moreover, this method of comparison is thought to provide some overview on how

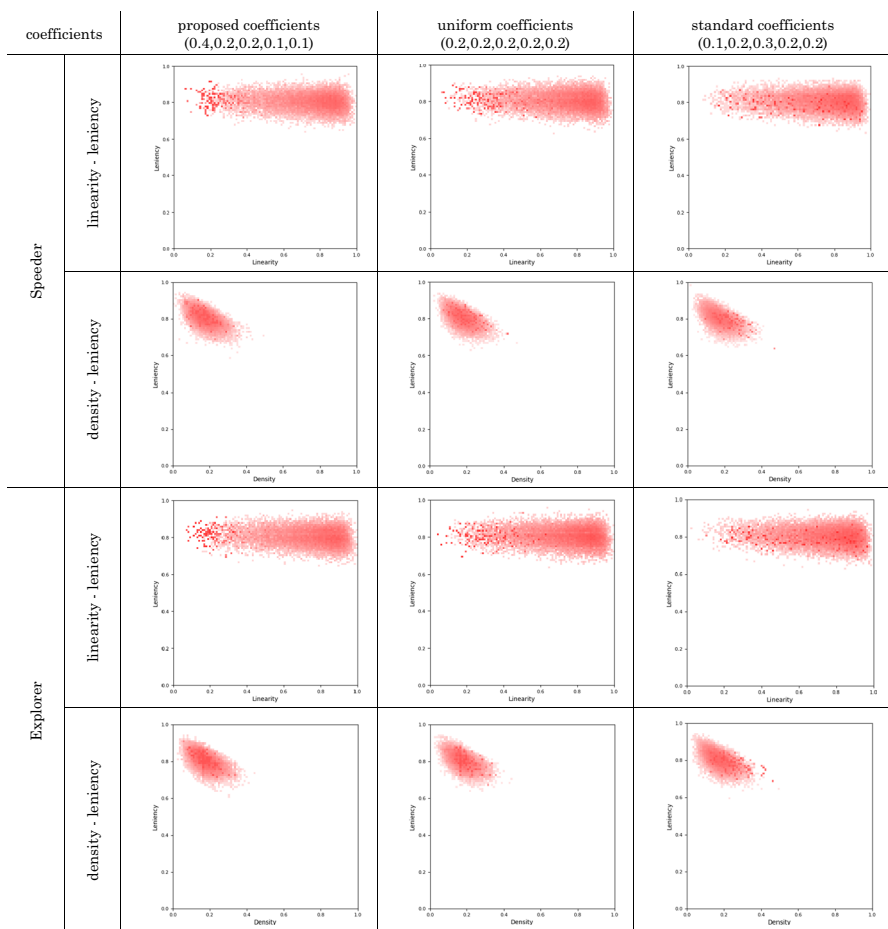


Fig. 9 Expressive range visualization of EDAPCG using all schematics and coefficients

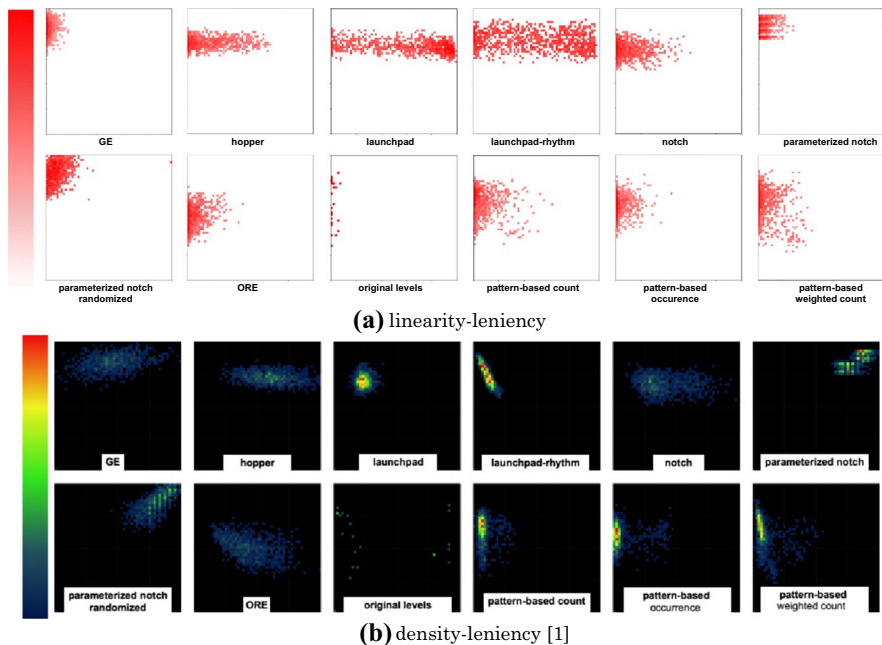


Fig. 10 ERA of previous methods—**a** linearity (x-axis) leniency (y-axis) reproduced using experimental data provided by B. Horn et al. [3], **b** density (x-axis) leniency (y-axis)

other methods perform, if they would have been configured towards EDAPCG's objectives.

Expressive range analysis

The Expressive Range Analysis (ERA) is a useful tool for comparing the capabilities of different methods in producing diverse content. First introduced by Smith et al. [22] for linearity-leniency, Horn et al. [3] also applied this tool for density-leniency analysis. As can be seen in the Fig. 9, compared to the expressive range of the other methods discussed in this study (Fig. 10), EDAPCG depicts an expressive range similar to both GE and pattern-based approaches (as it is designed to be a combination of both), and also to some level, hopper, notch and ORE except less diversity on leniency. Overall, EDAPCG expands the GE's expressive range by including more linearity values and significantly changing the range of pattern-based methods by shifting the focus of linearity in the cost of more lenient levels (Fig. 10, linearity – leniency).

In detail, EDAPCG's ERA for density metric is similar to Launchpad, Launchpad-rhythm and pattern-based variants, as they all generate levels with medium to low density values. For leniency metric, the output is somewhat limited and focused on certain areas and the majority of levels have moderate to high leniency. One possible cause of this behaviour can be attributed to the pre-defined schematics that limit certain DE usage that inherently prevents the leniency metric to drop to lower values (harder levels). In comparison with the previous methods, EDAPCG's

leniency behaviour is similar to most generators (higher values), except ORE and pattern-based approaches, which cover more values on the leniency spectrum. In terms of linearity, EDAPCG covers the majority of values, similar to Launchpad and hopper, improving on GE, pattern-based and ORE.

From another perspective, as can be seen from the expressive range of parameterized approaches (i.e. notch, Launchpad and hopper) and methodologies that incorporate various heuristics to increase pattern replication (i.e. Launchpad-rhythm and pattern-based), and to increase the controllability of the approach, usually the coverage of respective expressive range is limited significantly. However, EDAPCG tackles this challenge by using relaxed evolutionary process and probabilistic model heuristics, preserving the diversity of generated levels. Objective function coefficients have the most impact in this regard, as the distribution of the desired solutions

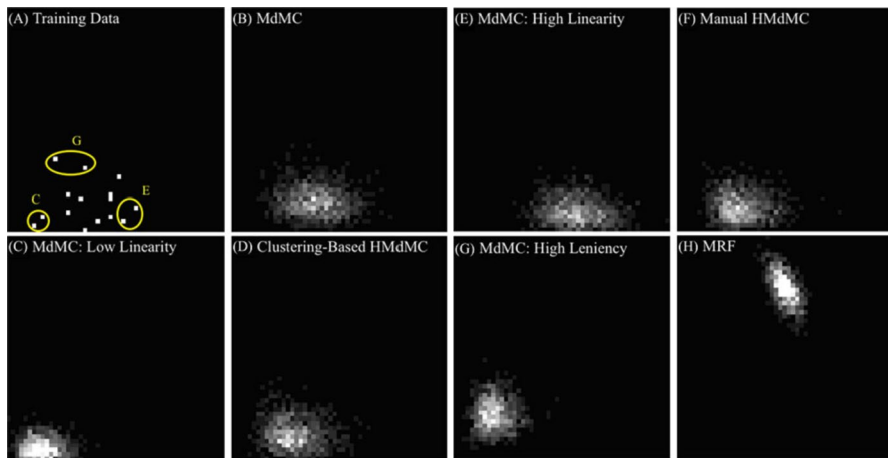


Fig. 11 Various expressive range visualization of S. Snodgrass et al. methods [39]

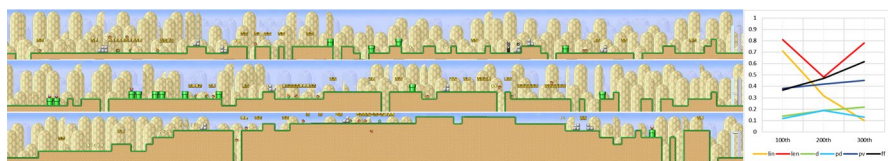


Fig. 12 Sampled levels with proposed coefficients (0.4, 0.2, 0.2, 0.1, 0.1), metric values (left to right: linearity, leniency, density, pattern density, pattern variation) and fitness function score—top, middle and bottom levels are 100th, 200th and 300th generations' elite individuals respectively

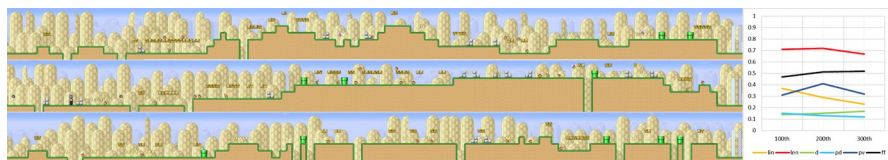


Fig. 13 Sampled levels with uniform coefficients (0.2, 0.2, 0.2, 0.2, 0.2)

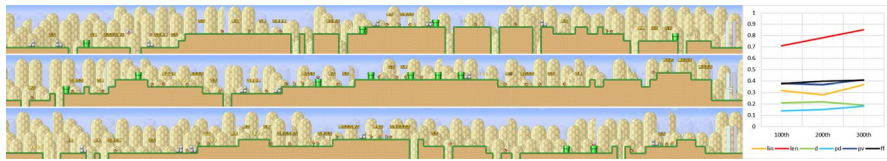


Fig. 14 Sampled levels with standard coefficients (0.1, 0.2, 0.3, 0.2, 0.2)

and the final output is more expanded in standard coefficients compared to the uniform and proposed counterparts, which hints toward the effectiveness of the underlying multi-objective evolutionary optimization design. It is also worth mentioning that the produced levels are tailored for two fixed and pre-defined playstyles, which inherently limits the evolutionary method's exploration. It is admissible to hypothesize that more adaptive playstyle categorization would yield improved expressive range coverage.

ERA comparison between EDAPCG and reinforced learning methods that incorporate probabilistic modeling is inherently less informative because the two do not quite belong to the same category. Many of the reinforced learning methods aim at high generalization between similar games agnostic towards domain knowledge. However, as can be seen in the Fig. 11, method introduced by S. Snodgrass et al. [38, 39] produces levels with similar expressive ranges (B-G) close to the samples used for learning and various probabilistic modeling structure (A).

Sample levels

In Figs. 12, 13 and 14 provided in this section, three representative generations are sampled from the EDAPCG's evolutionary process using different sets of coefficients to inquire about its workings. These sampled levels are the elite individuals from the 100th, 200th and 300th generations, with the latter being the final output of the level generation system. In addition, metric values for each level and fitness score are also included. It is worth mentioning that the showcased levels are not the comprehensive representatives of EDAPCG's expressive range, and are included for facilitating descriptive and visual discussion.

As can be seen in Fig. 12, the proposed coefficients tend to guide the evolutionary process towards optimizing the linearity metric over others. The justification for this design is the inherent ability of EDAPCG in producing patterns from the original game, which in turn increases the density and provides varied levels of leniency (usually high), allowing to shift focus to generate less linear levels. In Fig. 12, from top to bottom, a gradual shift towards less linearity can be observed. Interestingly, the middle level in Fig. 12 displays a rare and less lenient level compared to overall results, with several PIPE_PIRANHA and long gaps towards the end, thus it can be deduced that EDAPCG has the capacity to produce more variations of leniency, dependent on fine tuning the system using coefficients and probabilistic model's sampling agenda. This notion is explored through more experiments, which are included later in this section.

Figure 13, showcases sample levels in which EDAPCG manages to generate an output with balanced optimization of all the metrics. Specifically, notice the

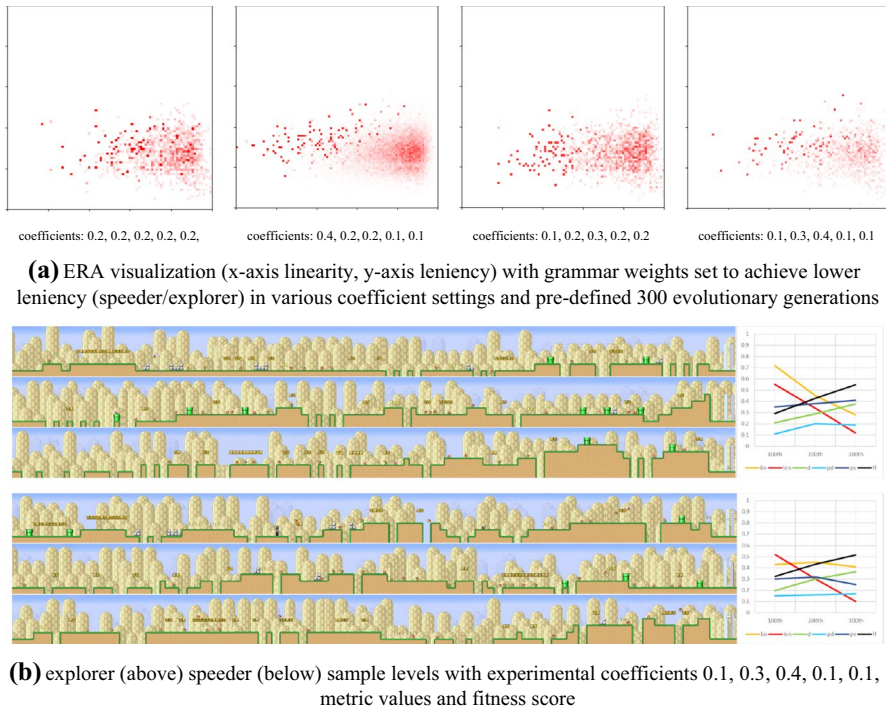


Fig. 15 Leniency experiments—**a** ERA visualization, **b** sample levels

lesser linear level is replaced by more density and pattern-oriented individual in 200th generation, giving way for a final output that satisfies all the metrics in a relatively balanced manner. By visual observation, the final output of this sample process is a speeder playstyle level, emphasizing on coins, jumps and sporadic enemies, whereas the previous level is more similar to an explorer type, with more enemy count and variation, powerup DE and less linearity. The evaluation for correct assigning of the final output of EDAPCG to target playstyles is included in designing of the system and informs the manually fine-tuning of different parameters for later runs.

In standard setting, the EDAPCG's goal is to increase the density and pattern-oriented metrics. The justification of this coefficient configuration can be visually deduced from Fig. 10—the original levels' expressive range. As can be seen in Fig. 14, the evolutionary process gradually moves towards more linear structure, while preserving density. However, from visual comparison between 200 and 300th elite individuals, some patterns and DE are discarded through the process (i.e. PIPE, STAIR), which may be due to EDAPCG's main priority to generate explorer type level, favoring jumps and coins to previously mentioned DE.

Upon further experiments in exploring more leniency values using the current system design, EDAPCG proves to be capable of achieving lower values (generating harder levels). The results of these additional experiments can be seen in Fig. 15. A new set of coefficients is introduced for this purpose. This objective function

setting is deduced empirically and has a relatively higher density metric coefficient, in which, EDAPCG can use various DE and specific patterns more freely to generate harder levels.

Lower leniency coefficients $\alpha_1=0.1$, $\alpha_2=0.3$, $\alpha_3=0.4$, $\alpha_4=0.1$, $\alpha_5=0.1$

By visual inspection of the ERA provided in Fig. 15a, for all coefficients, it can be easily gathered that EDAPCG achieves lower leniency levels and has more exploration in leniency compared to the previous experimental settings. The grammar rules contained in the schematics are not changed; however, the respective weights are all reset to uniform values (rendering them interchangeable) with emphasis on rules that result in generating harder levels (i.e. gap and enemy patterns and their respective DE are assigned maximum weights).

According to the sample levels for this experiment (Fig. 15b), gap and enemy patterns are very common. In detail, longer gaps and harder enemies (flying Goomba and SPIKY/Spiked red turtle) are prevalent. In addition, two and more hordes of enemies are more common. In terms of calculating leniency, more combinations of gaps decrease the leniency threefold, and the mentioned enemy sprites are considered twice as hard. Moreover extended gaps followed by enemy sprites and 2-horde of SPIKY can be seen in both sample levels (Fig. 15b). Regarding other metric values (Fig. 15b), linearity repeats its normal behaviour in reaching previously discussed diversity. Similarly, the pattern density metric remains unchanged, while density is increased notably and pattern variation is slightly limited due to the hard bias in producing leniency-reducing patterns.

In summary, as it is described by ERA and sample levels, EDAPCG can produce variation of levels to meet distinct playstyles and the evidence of effective PCG and controllability can be deduced based on the provided information. However, there are certain limitations and points of discussion, which will be covered in the next section.

4.2 Discussion

The performance of EDA for PCG on the IMB framework is reported for the first time in this paper. The main justification of use is the potential of probabilistic modeling unique to EDA, together with the compatibility for discrete optimization, possibility of formulating multi-objective evaluation function and direct mapping between phenotype and genotype. In addition, adding the principles of previous grammar-based approaches enables EDAPCG to reinforce the replication of meso-patterns of SMB. This combination of functionalities allows the proposed method to draw from the capabilities of previous methods and address their weaknesses, while bringing novel improvements to the topic of PCG in platformer games. A condensed overview of EDAPCG's specifications compared to other methods discussed in related works can be viewed in Table 13.

EDAPCG performs relatively modest in linearity and density metrics mainly because of the inherent limitations in its design. Specifically, EDAPCG lacks the platform DE that is used in some of the early methods, but omitted in the latest methodologies (e.g. [38–40]). Regardless, the proposed method performs relatively

Table 13 The specifications of EDAPCG and previous methods

| Content generator [ref.] | GE [3] | Hopper [31] | Launchpad [16] | Notch [31] | Notch param. [31] | ORE [31] | Pattern based [29] | Ben weber [31] | EDAPCG |
|-------------------------------------------------------------------------------|-----------------|-------------|----------------|------------------|-------------------|----------|--------------------|----------------|-----------------------------------|
| Probabilistic modeling based on problem and the course of finding the optimum | No | No | No | No | No | No | No | No | Yes |
| Design | | | | | | | | | |
| Evolutionary | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Multi-pass | | | | | | | | | |
| Evaluation function | Multi objective | – | – | Single objective | Single objective | – | Single objective | – | Multi objective |
| Metric priority | | | | | | | | | |
| Linearity | ✓ | | | | | ✓ | ✓ | | ✓ |
| Leniency | ✓ | | ✓ | ✓ | ✓ | ✓ | | | ✓ |
| Density | | ✓ | | ✓ | ✓ | | | | ✓ |
| Pattern density | | | | | | | ✓ | | ✓ |
| Pattern variation | | ✓ | | | | | ✓ | | ✓ |
| Compression distance | | | | | ✓ | | | | ✓ |
| Content generation | | | | | | | | | |
| Structural generate and test | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ |
| Online | | | | | ✓ | | | ✓ | |
| Offline | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | Yes | ✓ |
| Modeling | Yes | Yes | Yes | No | No | Yes | No | | Exploits built-in features of IMB |
| Direct | | | | | | | | | |
| Indirect | ✓ | ✓ | ✓ | | | ✓ | | ✓ | |

Table 13 (continued)

| Content generator [ref.] | GE [3] | Hopper [31] | Launchpad [16] | Notch [31] | Notch param. [31] | ORE [31] | Pattern based [29] | Ben weber [31] | EDAPCG |
|----------------------------|--------|-------------|----------------|------------|-------------------|----------|--------------------|----------------|--------|
| Evaluation | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ |
| Direct and theory driven | | | | | | | | | |
| Direct and data driven | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Phenotype-genotype mapping | | | | | | | | | |
| Direct | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Indirect | ✓ | | | | | | | | |

well in these aspects considering this limitation. In leniency metric, the proposed method shows good controllability, but has limited range compared to other methods. This shortcoming might be linked to the focus on controllability and the fact that EDAPCG uses predefined and static player models. These two factors limit the proposed method's tendency to produce a wide range of levels with different leniency values. However, according to the additional experiments, EDAPCG can be configured to explore lower leniency metric values (Fig. 15).

EDAPCG performs well in pattern-oriented metrics, but does not manage to outperform any of the pattern-based methods. This could be due to the proposed coefficient configurations in the objective function, which aim to improve all of the metrics, opposed to exclusively focusing on the meso-pattern replication designed in pattern-based methods. Moreover, estimation of distribution functionality in the probabilistic modeling of promising candidates and the evolutionary features enables EDAPCG (as a combination of grammar and pattern based method) to attain interesting results in compression distance metric, a unique advantage that requires more in-depth studies. In terms of technical aspects, EDAPCG generates playable levels without any major issue or delays; however, performance of the proposed method in this regard should be assessed and reported in the context of a sophisticated PCG competition.

Limitations

Parametric probabilistic modeling used in EDAPCG is agnostic to the structural relations between DE, and can be perceived as a basic form of modeling for a structural oriented PCG task such as level generation. Nevertheless, when perceived as a component in EDAPCG, which has sophisticated grammar rules for replicating such relations, the parametric probabilistic modeling provides a suitable form of automation in informing the evolutionary process on the previous generations, and is capable of providing statistical information for human author's investigations. In addition, compared to higher-order Markov chains and graphical probabilistic modeling techniques, it has low cost and can be perceived as suitable for similar applications in more complex and multi-faceted systems.

As can be deduced from the ERA and sample levels, the use of static and predefined player models, boiled down to only two categories, considerably limits the variation of EDAPCG expressive range. In addition, these rigid models often collide with the diversity of generated levels and guide them towards a certain expressive range (i.e. higher leniency). Moreover, EDAPCG, at its current stage and design goal, is not suitable for online content generation. In order to achieve this functionality, the probabilistic models can be trained before-hand similar to S. Snodgrass et al. [38, 39] and used in an online adaptive process. Compared to the parametric probabilistic modeling used in EDAPCG, higher-order Markov chains used in this method are costly to train, but greatly outperform the proposed method in level generation (EDAPCG takes around 7 min to generate level) with generating content in matter of milliseconds in MdMC and HMdMC models, and approximate 4 min for MRF model.

Compared with other approaches, the style-encoding probabilistic model introduced by Guzdial et al. [40] effectively captures level design style of original levels

via unsupervised learning from gameplay videos with no annotations. The evaluation for this method is based on online user study, outperforming the previously discussed method. EDAPCG uses predefined patterns as means to replicate level design style, thus is highly domain dependent with low generalizability to other genres. In addition, similar to CMA-ES, EDAPCG converges rather quickly to elite individuals, a limitation that is best reflected in empirically selected 300 evolutionary generation count. Moreover, for a conventional optimization method coupled with parametric probabilistic modeling and fixed playstyle categories, based on ERA, EDAPCG explores expanded areas of the solution space, which is reflected in top compression distance performance. Nevertheless, in no way EDAPCG can be compared to QD methods such as ME or CMA-ME, which are designed for maximal explorations [8]. Needless to say, EDAPCG in its current design adapts to player level of skill mainly through constraints, and the probabilistic models are used to provide overall overview of the PCG process, which is greatly limited when compared with fast and efficient methods introduced by Gonzalez-Duque 2021 [41].

5 Conclusion and future work

This paper is the first study on the performance of UMDA in PCG for game levels. EDA variants such as UMDA have roots in statistical and analytical sciences, and based on these principles, the hybrid proposed EDAPCG method provides relatively consistent results compared to previous methods. According to Tables 10–12, ERA and sample levels comparisons, an overall optimization in all of the target metrics results in an acceptable performance for EDAPCG. An interesting improvement is also achieved in compression distance metric compared to other evolutionary methods. Effective controllability using adjustable grammar rules and multi-objective evaluation function coupled with probabilistic models are the other note-worthy advantages of the proposed method. Providing high-level knowledge of the problem domain, as an alternative for feature selection limitations, and introducing some level of automation are other noticeable attributes of EDAPCG. In addition, direct phenotype to genotype mapping, generalization potential of the probabilistic modeling scheme and more variation in DE application are also mentionable.

Overall, this paper introduces the idea of parametric probabilistic/statistical modeling in the PCG domain and the results show high potential in this research direction. Structural modeling provides analytical information while creating a unique analysis over the problem domain. However, acquiring more in-depth and comprehensive information is not available through parametric modeling. Since the probabilistic modeling in this paper is parametric, thus the next step for future studies can be applying other structures of probabilistic modeling. In addition, a more advanced form of PEM can also be used in EDAPCG, creating a potential to mix player preferences adaptation and probabilistic modeling, which can yield interesting outcomes. Furthermore, applying the idea of probabilistic and structural modeling into other domains of AI in games can be an interesting outlook.

Appendix A. Design elements

| Flat surface | FLAT | Start and end of string | START, END | |
|-----------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------|-----------------------------|--|
| <i>coin</i> | COINS | <i>Block variations and combinations</i> | BLOCK_PP | |
| <i>pipe</i> | PIPE | | BLOCK_CC | |
| <i>piranha</i> | PIPE_PIRANHA | | BLOCK_EE | |
| <i>canon</i> | CANNON | | BLOCK_PC | |
| | | | BLOCK_PE | |
| | | | BLOCK_CE | |
| <i>gap</i> | GAP | <i>Variety of enemies (most of the methods limit the enemy type)</i> | GOOMBA | |
| <i>Increase in ground level</i> | GROUND_UP | | REDTURTLE | |
| <i>Decrease in ground level</i> | GROUND_DOWN | | GREENTUR- TLE | |
| <i>Stairs going up</i> | STAIRS_UP | | SPIKY | |
| <i>Stairs going down</i> | STAIRS_DOWN | | GOOMBA_ WINGED | |
| | | | REDTUR- TLE_ WINGED | |
| | | | GREENTUR- TLE_ WINGED | |
| | | | SPIKY_ WINGED | |
| | | | | |
| | | | | |
| <i>Variations and combinations of various blocks to add variety to patterns</i> | GOOMBA_BLOCK_PP, GOOMBA_BLOCK_CC, GOOMBA_ BLOCK_EE GOOMBA_BLOCK_PC, GOOMBA_BLOCK_PE, GOOMBA_ BLOCK_CE BLOCK_EE_GREENTURTLE, REDTURTLE_BLOCK_PE, GREENTURTLE_BLOCK_PP BLOCK_PC_GREENTURTLE, REDTURTLE_BLOCK_CE, GREENTURTLE_BLOCK_CC BLOCK_PE_GREENTURTLE, BLOCK_PP_GOOMBA, GREENTURTLE_BLOCK_EE BLOCK_CE_GREENTURTLE, BLOCK_CC_GOOMBA, GREENTURTLE_BLOCK_PC BLOCK_PP_REDTURTLE, BLOCK_EE_GOOMBA, GREENTUR- TLE_BLOCK_PE BLOCK_CC_REDTURTLE, BLOCK_PC_GOOMBA, GREENTUR- TLE_BLOCK_CE BLOCK_EE_REDTURTLE, BLOCK_PE_GOOMBA, REDTUR- TLE_BLOCK_PP BLOCK_PC_REDTURTLE, BLOCK_CE_GOOMBA, REDTUR- TLE_BLOCK_CC BLOCK_PE_REDTURTLE, BLOCK_PP_GREENTURTLE, REDTURTLE_BLOCK_EE BLOCK_CE_REDTURTLE, BLOCK_CC_GREENTURTLE, REDTURTLE_BLOCK_PC | | | |

References

1. G.N. Yannakakis, J. Togelius, Experience-driven procedural content generation. *IEEE Trans. Affect. Comput.* **2**(3), 147–161 (2011)
2. H. Mühlenbein, G. Paass. From recombination of genes to the estimation of distributions I. Binary parameters, in *International conference on parallel problem solving from nature*, pp. 178–187. Springer, Berlin, Heidelberg, (1996)
3. B. Horn, S. Dahlsgog, N. Shaker, G. Smith, J. Togelius, A comparative evaluation of procedural level generators in the Mario Ai framework, in *Foundations of digital games 2014. Society for the Advancement of the Science of Digital Games*, (2014)
4. J. Togelius, G.N. Yannakakis, K.O. Stanley, C. Browne, Search-based procedural content generation: a taxonomy and survey. *IEEE Trans. Comput. Intell. AI Games* **3**(3), 172–186 (2011)
5. J. Togelius, R. De Nardi, S. M. Lucas, Towards automatic personalised content creation for racing games, in *IEEE symposium on computational intelligence and games*, pp. 252–259, (2007)
6. A.M. Smith, M. Mateas, Answer set programming for procedural content generation: a design space approach. *IEEE Trans. Comput. Intell. AI Games* **3**(3), 187–200 (2011)
7. N. Shaker, M. Nicolau, G. N. Yannakakis, J. Togelius, M. O’neill, Evolving levels for super Mario bros using grammatical evolution, in *IEEE conference on computational intelligence and games (CIG)*, pp. 304–311, (2012)
8. M. C. Fontaine, J. Togelius, S. Nikolaidis, A. K. Hoover. Covariance matrix adaptation for the rapid illumination of behavior space, in *Proceedings of the 2020 genetic and evolutionary computation conference*, pp. 94–102. (2020)
9. D. Charles, M. Black, Dynamic player modeling: a framework for player-centered digital games, in *Proceedings of the international conference on computer games: artificial intelligence, design and education*, pp. 29–35, (2004)
10. G.N. Yannakakis, J. Hallam, Evolving opponents for interesting interactive computer games. *From Anim. Animats* **8**, 499–508 (2004)
11. G. N. Yannakakis, M. Maragoudakis, Player modeling impact on Player’s entertainment in computer games, in " *International conference on user modeling*, Springer, Berlin, Heidelberg, pp. 74–78, (2005)
12. G.N. Yannakakis, J. Hallam, Towards optimizing entertainment in computer games. *Appl. Artif. Intell.* **21**(10), 933–971 (2007)
13. G.N. Yannakakis, J. Hallam, Real-time game adaptation for optimizing player satisfaction. *IEEE Trans. Comput. Intell. AI Games* **1**(2), 121–133 (2009)
14. G.N. Yannakakis, M. Maragoudakis, J. Hallam, Preference learning for cognitive modeling: a case study on entertainment preferences. *IEEE Trans. Syst. Man Cybern.-Part A Syst. Humans* **39**(6), 1165–1175 (2009)
15. C. Pedersen, J. Togelius, G. N. Yannakakis, Modeling player experience in super Mario bros, in *IEEE symposium on computational intelligence and games*, pp. 132–139, (2009)
16. C. Pedersen, J. Togelius, G.N. Yannakakis, Modeling player experience for content creation. *IEEE Trans. Comput. Intell. AI Games* **2**(1), 54–67 (2010)
17. N. Shaker, G. N. Yannakakis, J. Togelius, Digging deeper into platform game level design: session size and sequential features, in *European conference on the applications of evolutionary computation*, Springer, Berlin, Heidelberg, pp. 275–284, (2012)
18. J. Togelius, R. De Nardi, S. M. Lucas, Making racing fun through player modeling and track evolution," (2006).
19. J. Togelius, J. Schmidhuber, An experiment in automatic game design, in *IEEE symposium on computational intelligence and games*, pp. 111–118, (2008).
20. T. W. Malone, What makes things fun to learn? Heuristics for designing instructional computer games, in *Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems*, pp. 162–169, (1980)
21. G. Smith, M. Treanor, J. Whitehead, M. Mateas, "Rhythm-based level generation for 2D Platformers, in *Proceedings of the 4th international conference on foundations of digital games*, pp. 175–182, (2009)
22. G. Smith, J. Whitehead, Analyzing the expressive range of a level generator, in *Proceedings of the 2010 workshop on procedural content generation in games*, p. 4, (2010)

23. M. Jennings-Teats, G. Smith, N. Wardrip-Fruin, Polymorph: a model for dynamic level generation, in Sixth artificial intelligence and interactive digital entertainment conference, (2010)
24. N. Sorenson, P. Pasquier, Towards a generic framework for automated video game level creation, in European conference on the applications of evolutionary computation, Springer, Berlin, Heidelberg, pp. 131–140, (2010)
25. N. Sorenson, P. Pasquier, The evolution of fun: automatic level design through challenge modeling, ICCG, pp. 258–267, (2010)
26. N. Sorenson, P. Pasquier, S. DiPaola, A generic approach to challenge modeling for the procedural creation of video game levels. *IEEE Trans. Comput. Intell. AI Games* **3**(3), 229–244 (2011)
27. M. Csikszentmihalyi, Toward a psychology of optimal experience, in *Flow and the foundations of positive psychology*, Springer, Dordrecht, pp. 209–226, (2014)
28. N. Shaker, G. N. Yannakakis, J. Togelius, Towards automatic personalized content generation for platform games, in Sixth artificial intelligence and interactive digital entertainment conference, (2010)
29. N. Shaker, G. N. Yannakakis, J. Togelius, Feature analysis for modeling game content quality, in IEEE conference on computational intelligence and games (CIG'11), pp. 126–133, (2011)
30. S. Dahlskog, J. Togelius, Procedural content generation using patterns as objectives, in European conference on the applications of evolutionary computation, Springer, Berlin, Heidelberg, pp. 325–336, (2014)
31. J. Roberts, K. Chen, Learning-based procedural content generation. *IEEE Trans. Comput. Intell. AI Games* **7**(1), 88–101 (2014)
32. N. Shaker, J. Togelius, G.N. Yannakakis, B. Weber, T. Shimizu, T. Hashiyama, N. Sorenson, P. Pasquier, P. Mawhorter, G. Takahashi, G. Smith, The 2010 Mario AI championship: level generation track. *IEEE Trans. Comput. Intell. AI Games* **3**(4), 332–347 (2011)
33. D. Perez-Liebana, S. Samothrakakis, J. Togelius, T. Schaul, S. M. Lucas, General video game AI: competition, challenges and opportunities, in Thirtieth AAAI conference on artificial intelligence, (2016)
34. D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, S. M. Lucas, General video game AI: a multi-track framework for evaluating agents, games and content generation algorithms, *IEEE Transactions on Games*, (2019)
35. T. Schaul, A video game description language for model-based or interactive learning, in IEEE conference on computational intelligence in games (CIG), pp. 1–8, (2013)
36. T. Schaul, An extensible description language for video games. *IEEE Trans. Comput. Intell. AI Games* **6**(4), 325–331 (2014)
37. J.K. Pugh, L.B. Soros, K.O. Stanley, Quality diversity: A new frontier for evolutionary computation. *Front. Robotics AI* **3**, 40 (2016)
38. S. Snodgrass, S. Ontañón. Experiments in map generation using Markov chains. FDG. 2014.
39. S. Snodgrass, S. Ontañón, Learning to generate video game maps using markov models. *IEEE Trans. Comput. Intell. AI Games* **9**(4), 410–422 (2017). <https://doi.org/10.1109/TCIAIG.2016.2623560>
40. M. Guzdial, M. O. Riedl, Game level generation from gameplay videos. *AIIDE 2016*: 44–50
41. M. González-Duque, R. B. Palm, D. Ha, S. Risi, Finding game levels with the right difficulty in a few trials through intelligent trial-and-error, in 2020 IEEE Conference on Games (CoG), 2020, pp. 503–510, <https://doi.org/10.1109/CoG47356.2020.9231548>
42. M. Hauschild, M. Pelikan, An introduction and survey of estimation of distribution algorithms. *Swarm Evol. Comput.* **1**(3), 111–128 (2011)
43. M. Pelikan, M. W. Hauschild, F. G. Lobo. Introduction to estimation of distribution algorithms. MEDAL Report 2012003 (2012).
44. A. Summerville, J. RH Mariño, S. Snodgrass, S. Ontañón, L. HS Lelis. Understanding mario: an evaluation of design metrics for platformers, in Proceedings of the 12th international conference on the foundations of digital games, pp. 1–10. (2017)