Generative Art via Grammatical Evolution

Erik M. Fredericks
School of Computing
Grand Valley State University
Allendale, Michigan
Email: frederer@gvsu.edu

Abigail C. Diller
School of Computing
Grand Valley State University
Allendale, Michigan
Email: dillerab@mail.gvsu.edu

Jared M. Moore
School of Computing
Grand Valley State University
Allendale, Michigan
Email: moorejar@gvsu.edu

***Abstract*—Generative art produces artistic output via algorithmic design. Common examples include flow fields, particle motion, and mathematical formula visualization. Typically an art piece is generated with the artist/programmer acting as a domain expert to create the final output. A large amount of effort is often spent manipulating and/or refining parameters or algorithms and observing the resulting changes in produced images. Small changes to parameters of the various techniques can substantially alter the final product. We present *GenerativeGI*, a proof of concept evolutionary framework for creating generative art based on an input suite of artistic techniques and desired aesthetic preferences for outputs. *GenerativeGI* encodes artistic techniques in a grammar, thereby enabling multiple techniques to be combined and optimized via a many-objective evolutionary algorithm. Specific combinations of evolutionary objectives can help refine outputs reflecting the aesthetic preferences of the designer. Experimental results indicate that *GenerativeGI* can successfully produce more visually complex outputs than those found by random search.**

***Index Terms*—generative art, evolutionary algorithms, grammatical evolution, genetic improvement**

I. Introduction

Generative art is produced via programming techniques, typically via algorithmic design or artificial intelligence [1]–[5].[1] Algorithmic techniques generally focus on visualization of some form of mathematical formula or data analysis technique and may be parameter-driven to fine-tune the output. For example, a sine wave can be visualized by iterating over pairs of calculated (x, y) coordinates and then a graphics library (e.g., Python - Pillow (PIL), p5.js, etc.) may be used to draw and color each point as desired. Figure 1 demonstrates the visualization of $y = 75.0 * sin(x)$.[2]

Visualization of algorithmic techniques generally falls within two categories: expressing behaviors/trends in data and/or algorithms or for artistic purposes [1], [2]. Often visual interest can be created when two or more techniques are combined. Many generative techniques are amenable to encoding as genes within an evolutionary process given that parameters can change the behavior of the technique and consequently the resulting image [4], [5]. For example, configurable parameters for Figure 1 include mathematical variables (e.g., amplitude, period, spacing between each point) as well as drawing parameters (e.g., palette, point size, etc). Grammatical evolution (GE) is a subset of genetic programming that represents its genome in a grammar-based format [6]. The nature of the grammar introduces inherent constraints within the genome, ideally reducing the amount of invalid individuals generated over the course of execution. Grammar-based approaches have also been recently used in genetic improvement applications [7]. As such, manual selection of generative techniques and their subsequent parameters can often be a time-consuming process to create an ideal output. Moreover, algorithmically assessing the "quality" of generated outputs can be difficult without human intervention [8].

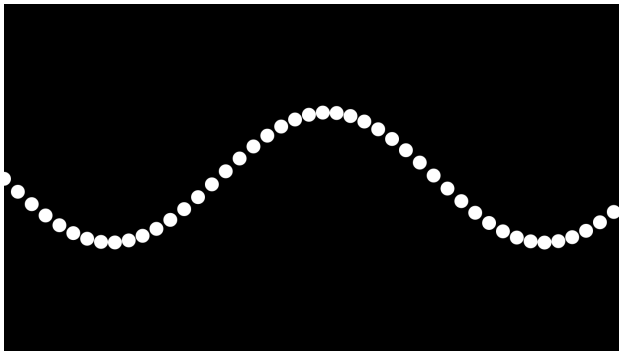


Fig. 1: Visualization of sine wave in p5.js.

We present *GenerativeGI*, a framework for creating generative art via GE. *GenerativeGI* takes as input a suite of parameterized generative techniques yielding a set of artistic outputs based around their evolved combination, with the intent being to generate new and interesting combinations of techniques and parameters. For example, Figure 1 may be combined with a *pixel sorting* technique (i.e., ordering pixels in an image based on colors, intensities, etc.) to generate a "glitch art" output as seen in Figure 2.

GenerativeGI comprises two main steps: grammar generation and evolution. First, a domain expert must translate a suite of generative art techniques to a grammar-based format. Specifically, we use the Tracery library [9] for lightweight grammar manipulation. A separately-callable function must be included for visualizing the technique within *GenerativeGI*. Next, the fitness function(s) for the evolutionary process must be configured based on desired output style, run time, etc. For example, a "glitch art" style can be accomplished by maxi-

We gratefully acknowledge support from the Michigan Space Grant Consortium (award 80NSSC20M0124) and Grand Valley State University.

[1]For the purposes of this paper we focus on algorithmic techniques.

[2]Based on https://p5js.org/examples/math-sine-wave.html.

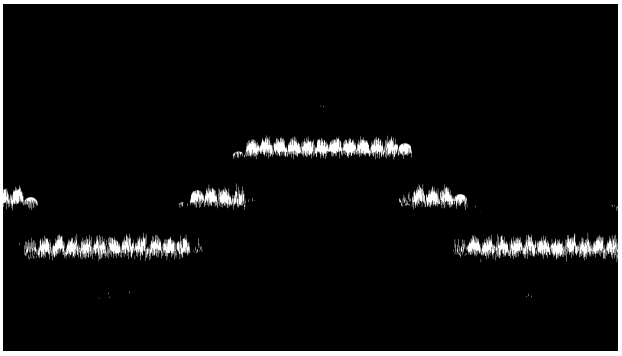


Fig. 2: Sine wave output combined with pixel sorting.

mizing the difference between pixel values of output images (e.g., inducing general *noise* in the output images). To manage competing concerns in fitness functions we use Lexicase, a many-objective approach for evolutionary selection [10].

Experimental results indicate that *GenerativeGI* can generate significantly more desirable (i.e., with respect to the aesthetic preferences of the designer and the specified fitness functions) output images than can be found via random search. The remainder of this paper is organized as follows. Section II presents relevant background information on generative art and grammatical evolution. Section III then details the *GenerativeGI* technique. Following, Section IV presents our experimental results and Section V highlights related work. Lastly, Section VI summarizes our findings and presents future research directions.

II. Background

A. Generative Art

Generative art is produced through functions often based on mathematical or computational principles that map to a visual medium, where outputs can include artistic endeavors and algorithm/data visualization techniques [1]–[5]. Introducing students to programming with generative art is an effective approach as the resulting visual products can provide fast and interesting feedback to those still learning (i.e., creative coding) [11]–[15]. Furthermore, generative techniques have also been applied to Internet of Things environments for expressing art in real-world settings [16].

Figure 3 presents a subset of the techniques used in this paper.[3] The dithering technique is also used in the genome but not shown in Figure 3 for presentation purposes. For this project each technique can act upon the entirety of the image canvas and will minimally accept an image object and color palette as input. Specifics of the techniques are described as follows:

Stipple: Stippling is a drawing technique that uses a large number of dots with varied spacing to create a texture or gradient effect. Parameters include the density and color of drawn points. Figure 3a illustrates this technique.

[3]Note: we enumerate each of the techniques used in our public repository: https://github.com/GI2023-GenerativeGI/GI2023.

Cellular Automata: A cellular automata is a rules-based technique for procedurally drawing shapes/figures. Cells are typically drawn based upon the state of their neighboring cells, with common examples including Conway's Game of Life and the Wolfram automata [17]. Figure 3b presents a sample based on the Wolfram ruleset.[4] Parameters include the colors for each cell as well as the ruleset itself.

Pixel Sorting: Pixel sorting orders the pixels of an image based upon a desired outcome, where sample outcomes include sorting based on hue, lightness, average red/green/blue (RGB) values, or other measures. Figure 2 presents a pixel sorted version of the sine wave shown in Figure 1. We use an open source Python module for pixel sorting.[5] All non file-based parameters available in the included pixel sorting module (e.g., sorting algorithm, thresholds, etc.) are used to specify its behaviors when acting upon an image.

Circle Packing: Circle packing is an algorithm for filling a space with as many circles as possible without overlap. Figure 3c presents a sample output. Its input is the number of attempts to place a circle (i.e., to avoid timing out if a suitable location cannot be easily found).

Flow Field: Flow fields are visualizations of vector fields that generally illustrate fluid motion, where a grid of values is instantiated based on an underlying noise function (e.g., Perlin [18], Simplex [19], etc.) governing the direction that particles and/or vertices may take as they traverse the grid [2], [3]. Figure 3d provides a sample flow field where the noise values are mapped between 0 and 2π. Parameters for the flow fields here include the type of noise mapping (i.e., flow style), noise resolution (i.e., zoom level), and number of particles.

Drunkard's Walk: The Drunkard's walk is an random algorithm in which an object probabilistically selects a direction to move each step of execution and then draws itself at that location. This technique can lead to interesting visual patterns (c.f., Figure 3e) and has been used for many applications, including physics and biological models [20] as well as procedural content generation [21].

Dithering: In image applications, dithering is typically used to provide the illusion of color depth and/or texture for images with a limited palette [22], [23]. These techniques often use a small image pattern to represent pixel values (e.g., shading, material textures, etc).[6]

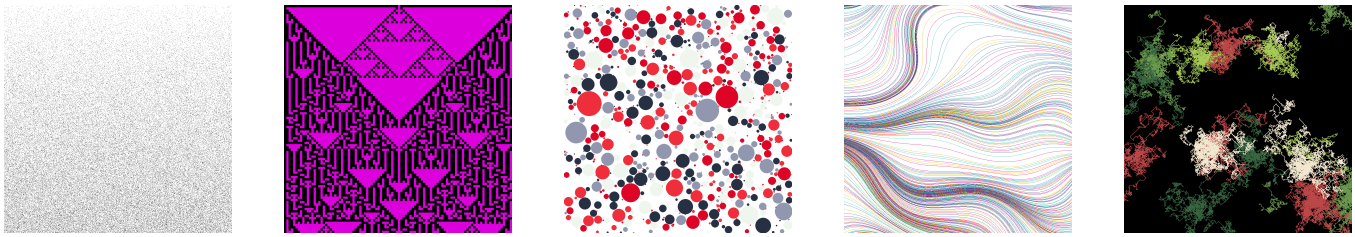
B. Grammatical Evolution

GE is a subset of evolutionary computation, specifically genetic programming [24], that searches for a solution using a *grammar*-based genome [6], [7]. A grammar-based approach enables the domain expert to constrain the space of input solutions to those defined within the rules of the genome and moreover can be a target for genetic improvement [7]. The remainder of the evolutionary operators (i.e., selection, mutation, crossover) are similar to those found in genetic

[4]See https://p5js.org/examples/simulate-wolfram-ca.html.

[5]See https://github.com/satyarth/pixelsort/.

[6]Note: this technique is not pictured as it is difficult to visualize when printed in monochrome in a small space.



(a) Stippled (b) Cellular Automata (c) Circle Packing (d) Flow Field (e) Drunkard's Walk

Fig. 3: Generative art techniques used as genomic elements in the evolutionary process.

algorithms and genetic programming. While any form of grammar representation is feasible with GE, we use the *Tracery* library [9] for ease of use.

Tracery: Tracery[7] is a framework for enabling procedural text generation in a grammar-based format [9]. Specifically, a grammar snippet in Tracery can be represented in a Python dictionary as follows (note: Tracery has been adapted to multiple programming languages):

```
rules = {
    'ordered_pattern': ['#techniques#'],
    'techniques': ['#technique#',
        '#techniques#,#technique#'],
    'technique': ['stippledBG',
        'flowField'],
    ...
}
```

Here, the name of the production is represented as a dictionary key (i.e., `ordered_pattern`, `techniques`, etc.) and can be expanded when placed between # symbols in the dictionary's value field. In this sample, `ordered_pattern` is the entry point to the grammar that is then expanded with the `techniques` production. `techniques` may either end with a specific `technique` or be expanded recursively (i.e., `#techniques#, #technique#`). If there are multiple options within a production, Tracery will select and return a random value. A sample *flattened* grammar (i.e., productions expanded in-code) is as follows (for the purposes of this example, we simplify the production to only show the technique name and abstract the numerous parameters that configure each technique):

```
stippledBG(params),stippledBG(params),
flowField(params),stippledBG(params),
```

GenerativeGI would accept this string as input and sequentially execute each specified technique (in this case, stippling twice, drawing a flow field, and then stippling once more).

Taking into account that each genome is of variable length, we use single-point crossover and single-point mutation to generate new individuals. For crossover, we select a random index into each parent and swap genomes at that point to generate children, ensuring that each governing technique (e.g., `stippledBG(`$params$`)`) remains intact.

Two forms of mutation can be applied at a random index. First, the technique at the selected index may be replaced with a new technique (or techniques, depending on the flattening of the grammar) (i.e., mut_1) significantly changing the generated output image. Second, the technique's parameters may be randomized (i.e., mut_2), thereby altering the appearance of the produced image specific to the referenced technique (as opposed to fully regenerating the grammar at that point). For example, assume that the technique at the selected index returned `flowField('edgy', 0.01)` (i.e., a flow field maps a noise function to "hard" angles and expresses a large amount of detail from its noise function). Flattening the grammar at that index could return a new technique or recursively return a set of techniques. For mut_1, the index could be replaced by `dither('halftone'), flowField('edgy', 0.025), dither('grayscale')`. In this case, flattening the grammar produced a recursive result that yielded three new techniques that were injected into the individual. For mut_2, the parameters for `flowField` can be randomized according to their specified valid ranges within the grammar.

C. Lexicase Selection

We use the Lexicase selection operator for selection events within the evolutionary process. Lexicase selection is a many-objective algorithm for searching through a large search space with multiple competing objectives [10]. Originally proposed for genetic programming, Lexicase selection has been applied to program synthesis [25], evolutionary robotics [26], [27], and geosciences [28], among others. In contrast to pareto-based multi-objective optimization approaches like NSGA-III [29], Lexicase evaluates a sample of individuals on an objective-by-objective basis. During each selection event, a sample of the population is taken along with a random shuffling of the objectives. A comparison of the sampled individuals is done based on performance in the first objective. If one individual is better than the rest of the sample it is selected. However, if two or more individuals are tied, the tied individuals advance to comparison based on the next objective and the process repeats. If all objectives are exhausted and there are still two or more individuals with similar performance, a random selection from the remaining individuals is used to choose an individual.

Considering the real-valued objectives that we use in this study, we employ ϵ-Lexicase selection [30], a variant that considers two individuals tied if the performance of an individual is within an ϵ of the max fitness individual. This is important in

[7]See http://www.crystalcodepalace.com/tracery.html.

real-valued fitness objectives as small differences in calculated fitness (e.g. distance traveled in a legged robot [31]) may not manifest as substantial changes to the observable output of evolved solutions. We use $\epsilon = 0.85$ in this study based upon empirical evidence.

III. Approach & Running Example

GenerativeGI is an evolutionary framework for enabling the creation of generative artwork via grammatical evolution and many-objective search. First, we present our motivating example and then discuss the specifics of *GenerativeGI*.

A. Motivating Example

For illustrative purposes, consider a *flow field* algorithm (c.f., Figure 3d) in which a vector field is generated via a noise function and mapped to a specified angle range [3]. One approach for implementing a flow field is to instantiate a list of particles (comprising at minimum (x, y) coordinates) that each draw a specified shape at their current position and then update their position based on flow field parameters (e.g., offset from current position, particle lifetime, etc.) and the current position's grid angle.

B. GenerativeGI Process

GenerativeGI creates images by *layering* the output from generative art techniques upon a virtual image canvas and therefore requires that each technique be implemented within our framework, where the selection, ordering, and instantiation of each technique is guided via the evolutionary process.

Figure 4 presents a data-flow diagram of *GenerativeGI* and we next describe each step in detail.

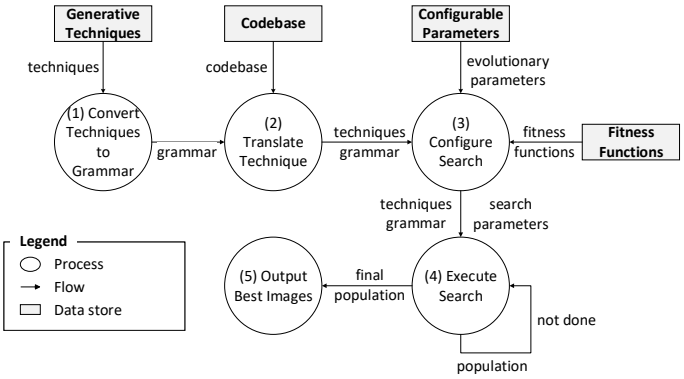


Fig. 4: Data flow diagram for *GenerativeGI*.

Step (1) - Convert Techniques to Grammar: *GenerativeGI* requires a suite of generative art techniques as input. As such, each technique must be self-contained, parameterized, and translated to a grammar production. A sample grammar conversion for a flow field is as follows in Listing 1.

```
rules = {
  ...
  'technique'  : ['flow-field', ...],
  'flow-field' : '#flow-field-type#:#palette#:#flow-
      field-zoom#',
  'flow-field-type' : ['edgy', 'curves'],
  'flow-field-zoom' : [str(x) for x in np.arange
      (0.001, 0.5, 0.001)],
  ...
}
```

Listing 1: Flow field grammar production.

A grammar generated based on Listing 1 will yield a random type of flow field and random zoom level when flattened (based on a pre-selected list of floating point values that are converted to strings per Tracery's requirements). Note: the grammar does not necessarily need to represent all possible parameters. For this example, the image, number of rows/columns, and color palette can be passed in manually or added to the grammar, as desired. In this case, the number of rows/columns match the image height/width, respectively. The remaining parameters would be passed based on the parsed grammar in Listing 1.

Step (2) - Translate Technique: Each included technique must be defined as a self-contained function that accepts minimally an image as input. Specifically, the function must be callable by *GenerativeGI* as the grammar is parsed and evaluated sequentially. Each function will overlay its output upon the passed image as specified by the generative technique.

Step (3) - Configure Search: Next, the evolutionary search process must be configured in terms of execution type (i.e., many-objective, single-objective, or random), number of generations to execute, population size, and the number of individuals to create via crossover/mutation. Given that each genome is of variable length we use single-point crossover and mutation, respectively (c.f., Section II-B). The fitness function(s) must also be defined to ensure that the search converges to a set of ideal outputs. For the purposes of this paper, we focused on minimizing the number of duplicate genes across individuals (i.e., $ff_{min(genome)}$), maximizing the diversity of the included techniques (i.e., $ff_{max(techniques)}$), and maximizing the differences between generated images (i.e., $ff_{max(RMS)}$ and $ff_{max(Chebyshev)}$). We use both a pairwise root mean square (RMS) and Chebyshev distance metric to measure the differences between pixel values for two images. For both distance metrics, we perform a pairwise comparison between an image and every other image in the current population and then average based on the largest discovered distance value, representing the overall novelty of that particular image with respect to the other members [32]. We further detail our evolutionary configuration in Section IV.

Step (4) - Execute Search: The evolutionary process is then executed according to its configured parameters. Each member of the population is evaluated per its flattened grammar, where each generative technique specified in the genome is executed sequentially on the individual's image.

GenerativeGI then executes for the configured number of generations and performs the crossover/mutation/selection operators for each generation. New individuals generated via the crossover and mutation operators retain the data within their image object, however their grammar is updated with respect to the operation performed. Selection of individuals is managed based upon the configured search technique, where

many-objective search uses Lexicase selection [10], single-objective search uses tournament selection, and no selection is necessary for random search.

Step (5) - Output Best Images: *GenerativeGI* will output the final population of images upon completion of the search and denote those considered to be "most fit" according to the configured fitness function criteria.

IV. Experimental Results

This section details our experimental configuration and results, respectively.

A. Experimental Configuration

For this paper, we translated a set of generative art techniques to use Python's Pillow library[8] for inclusion in *GenerativeGI*. Each technique was then encoded to a Tracery grammar comprising the name of the technique, configuration parameters, and specifications for valid ranges/values for the parameters. Overall, eight techniques are targets for evolution in this paper: *circle packing*, *drunkard's walk*, *stippling*, *dithering*, *pixel sorting*, *cellular automata*, and *flow field* (two separate implementations). The specifics of each technique were briefly described in Section II-A.

An individual's grammar, comprising a suite of techniques and instantiated parameters, is then expressed sequentially while drawing on its image object, where the image object is not cleared during evolutionary crossover and mutation. We intentionally preserve the parent's drawing canvas (i.e., image object) to provide more aesthetically-interesting outputs. Additionally, the execution time for each individual technique may vary due to the complexity of the technique itself (e.g., each individual pixel may need to be iterated over multiple times).

We compare and contrast three experimental treatments: many-objective search (i.e., Lexicase selection using $ff_{max(RMS)}$, $ff_{min(genome)}$, $ff_{max(techniques)}$, and $ff_{max(Chebyshev)}$), single-objective search (i.e., only using $ff_{max(RMS)}$ for evaluation), and an equivalent amount of randomly-generated solutions [33]. To optimize processing time for the random treatment we performed 50 evaluation cycles, each with 100 randomly-generated solutions. For statistical evaluation we performed 10 experimental replicates for each treatment.

Parameter	Value
Experimental replicates	10
Image size (pixels)	1000 x 1000
Number of generative techniques	8
Generations	100
Population size	100
Crossover rate	0.5
Mutation rate	0.4
Number of Lexicase objectives	4
ϵ (Lexicase - many-objective)	0.85

TABLE I: Evolutionary parameter configuration.

[8]See https://python-pillow.org/.

B. Empirical Evaluation

Figure 5 presents boxplots of the novelty scores for the "most novel" individual for each replicate between experimental treatments (i.e., Lexicase, single objective, and random). We compare the treatments against each other using the Wilcoxon rank-sum test with Bonferroni correction. The novelty scores for Lexicase versus random are statistically significant ($p < 0.01$) as is single-objective versus random ($p < 0.03$).

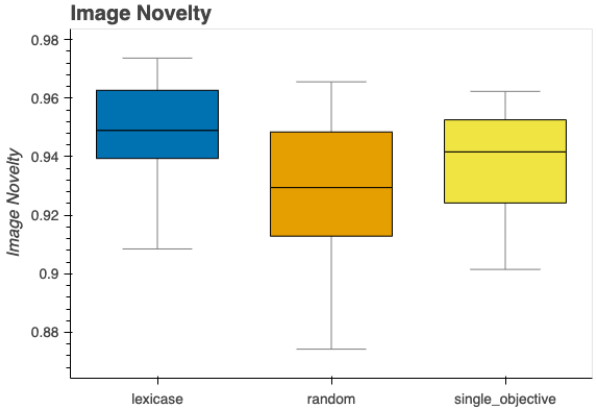


Fig. 5: Novelty score for the most novel individual per replicate across treatments.

Figure 6 shows the mean of the population's gene uniqueness scores (i.e., $ff_{min(genome)}$ for each replicate between experimental treatments. This minimization objective represents diversity within our population as its value reflects the number of other individuals in the population with the same gene. Lower scores indicate a more diverse population. Random has such a low score relative to Lexicase and single objective as the individuals are generated randomly and not evolved, therefore it is rare that many duplicate genes exist. Random is significantly different to Lexicase and single objective with $p < 0.001$ for each. Lexicase is significantly different ($p < 0.001$) than single objective indicating that it is better at maintaining a diverse population producing a wider variety of artistic outputs.

Figure 7 presents a set of images generated from the most fit individuals from each experimental treatment, where the left column shows images from many-objective search (i.e., Lexicase selection), the middle column shows random search, and the right column shows single-objective search.

Both single- and many-objective search tended to converge to multiple techniques in the resulting image, whereas random search tended to have less techniques with more "blank space" within its outputs. This convergence mainly stems from our fitness goals of maximizing pixel differences between images, thereby encouraging more distinctive features. This outcome will also result in less "negative space" as a byproduct, where negative space may be a feature of interest (in which case, an additional fitness function would need to be introduced to pressure the evolutionary process). While these results

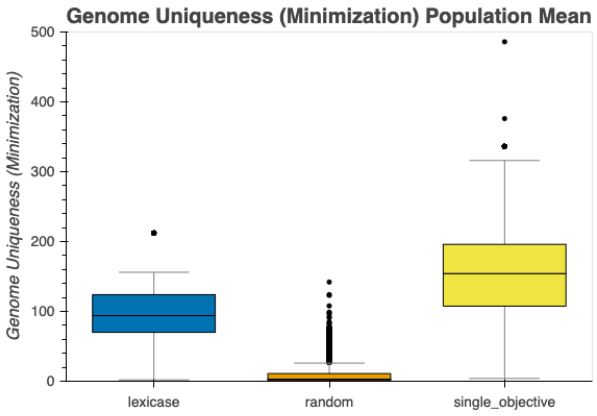


Fig. 6: Population mean genome uniqueness score per replicate across treatments. Lower values indicate a more diverse population.

are encouraging for a proof of concept study, quantifying "aesthetics" within an environment is a non-trivial problem and as such is a target for future study and is discussed within Section V.

We included Lexicase selection to enable many-objective search, given the large possible search space of this application (i.e., number of techniques, number of possible parameter values, etc.). For this proof of concept experiment we only included four objectives. From prior experiments, however, Lexicase tends to excel with a larger number of fitness functions [34]. As future work, we intend to significantly expand our suite of fitness functions to further expand the artistic possibilities of the software.

Additionally, as outputs were generated (per replicate) we noted that Lexicase selection tended to converge towards a set of common outputs, where those outputs were generally governed by a small set of generative techniques particular to that replicate (e.g., flow field and pixel sorting, stippling and cellular automata, etc.). While such a result may be desirable given the evolutionary process (i.e., to converge to a common solution) we also understand that a set of artistic outputs ideally would be relatively distinct from each other to enable the generative artist a wide range of diverse choices. To mitigate these concerns we would further propose expanding our fitness functions and suite of generative techniques as future work.

Threats to Validity: This research is intended as a proof-of-concept study to determine the feasibility of directing the creation of generative art via grammar-based optimization. One threat to validity lies in the derivation and validity of the fitness functions as we were mainly interested in maximizing difference (i.e., yielding "glitch art"). Another threat to validity lies in the image sizes, as we set them to be 1000 x 1000 pixels in size (a relatively small image in comparison to the larger resolutions found in traditional digital artwork) and therefore have a reduced resolution to draw upon and compare/evaluate. A third threat lies in the small number of art techniques as we mainly applied techniques with which we were familiar. An additional threat lies in the artistic merit of these outputs with respect to formal art theory.

V. Related Work

This section highlights relevant related work on creating generative art via artificial intelligence (AI) techniques and other search heuristics.

A. Generative Art via Artificial Intelligence

AI-driven art has recently exploded in popularity, with current techniques including diffusion-based techniques (e.g., DALL-E, Midjourney, Stable Diffusion, etc.) and VQGAN-CLIP [15], [35], [36]. Such techniques are typically rooted in deep learning approaches (such as GPT-3 [37]) that leverage a massive suite of source images and are prompt-driven (i.e., based on text input from users and then iterated/configured as needed). While such techniques are quite powerful and can result in relevant outputs, they require both a massive dataset and large amounts of computing capabilities. In contrast, *GenerativeGI* only requires a suite of input generative techniques and the computing power to execute evolutionary search.

B. Generative Art via Search Heuristics

Liu and Liu leverage a tree-based genetic algorithm, supported by a human operator for evaluation as desired, to visualize 3D models of mathematical formulae [4]. De Smedt *et al.* use an existing generative art framework, NodeBox, for use in an evolutionary computation-driven application for creating environments for game worlds [5]. While each of these are exploring a similar domain, *GenerativeGI* provides a grammar-based approach that enables fine-tuned control over the input drawing techniques.

Johnson also provides a taxonomy of fitness metrics for evolutionary art/music, where such metrics are a non-trivial problem as human preferences must be considered in the outputs [8]. This taxonomy in particular can serve as an excellent resource for future studies in which different fitness functions are evaluated within our framework.

VI. Discussion

This paper has presented *GenerativeGI*, a framework for optimizing grammars with the intent of creating generative artwork. We provided a proof-of-concept empirical evaluation to demonstrate its effectiveness in comparison to random search. Experimental results suggest that guided optimization can provide far more interesting and aesthetically pleasing outputs than can be found randomly.

This project initially started as an approach for automatically creating and generating artwork via algorithmic techniques, however the research potential for this project is quite vast. Specifically, we anticipate near-term future directions for this project to include exploring different forms of fitness functions (especially those proposed by Johnson [8]), incorporating human feedback within the evolutionary process, and augmenting the classification and comparison of outputs to provide a more intertwined and natural result.

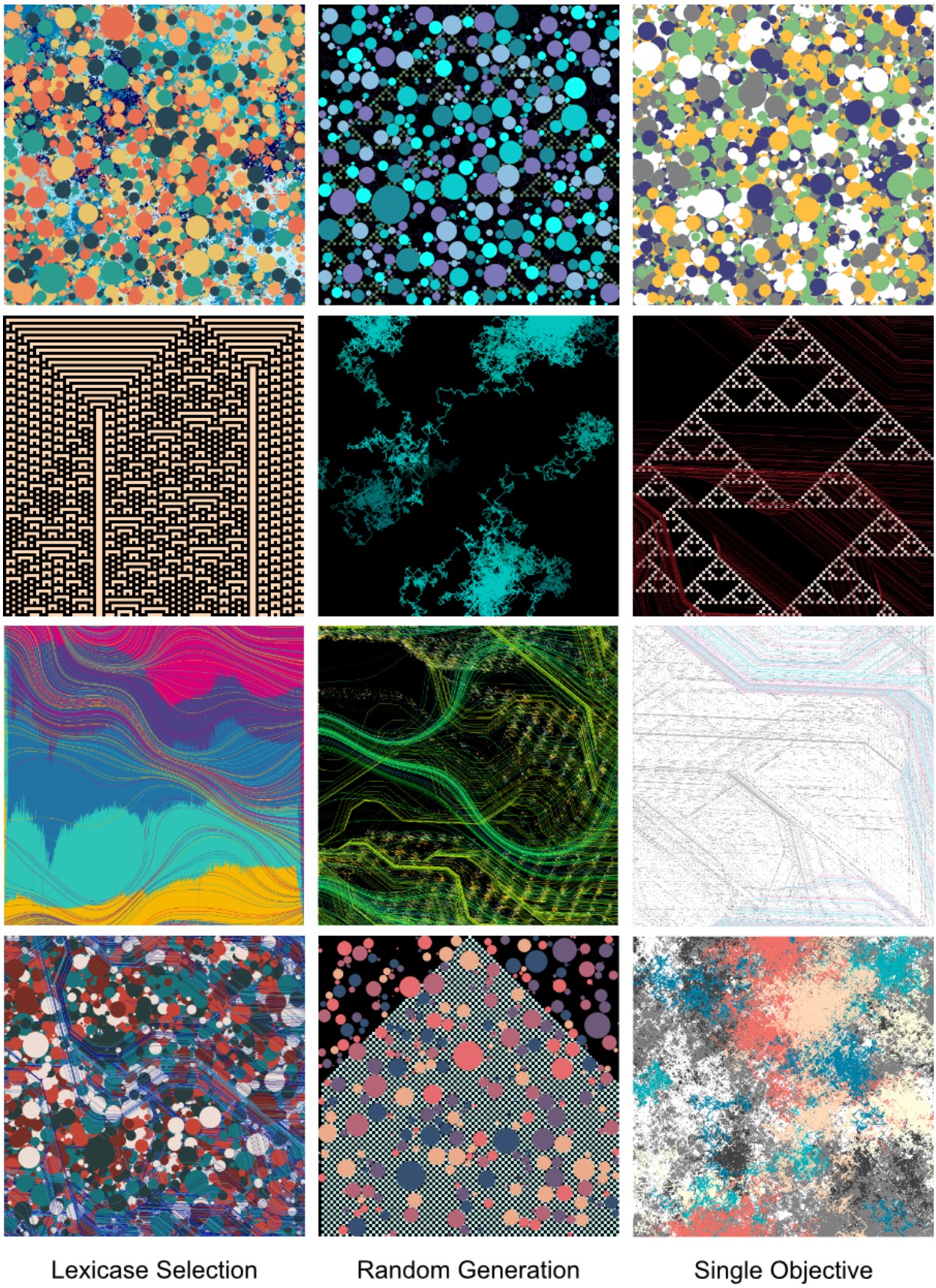


Fig. 7: A sample of the images generated across treatments. Lexicase Selection and Single Objective are individuals that have evolved over 100 generations while Random Generation represents the random initial state generated by the grammar.

References

[1] M. A. Boden and E. A. Edmonds, "What is generative art?" *Digital Creativity*, vol. 20, no. 1-2, pp. 21–46, 2009.

[2] A. G. Forbes, T. Höllerer, and G. Legrady, "Generative fluid profiles for interactive media arts projects," in *Proceedings of the Symposium on Computational Aesthetics*, 2013, pp. 37–43.

[3] B. Cabral and L. C. Leedom, "Imaging vector fields using line integral convolution," in *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, 1993, pp. 263–270.

[4] H. Liu and X. Liu, "Generative art images by complex functions based genetic algorithm," *Trends in Computer Aided Innovation*, pp. 125–134, 2007.

[5] T. D. Smedt, L. Lechat, and W. Daelemans, "Generative art inspired by nature, using nodebox," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2011, pp. 264–272.

[6] C. Ryan, J. Collins, and M. O. Neill, "Grammatical evolution: Evolving programs for an arbitrary language," in *Genetic Programming*. Springer, 1998, pp. 83–96.

[7] W. B. Langdon, "Genetic improvement of genetic programming," in *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2020, pp. 1–8.

[8] C. G. Johnson, "Fitness in evolutionary art and music: a taxonomy and future prospects," *Fitness in evolutionary art and music: a taxonomy and future prospects*, vol. 9, no. 1, pp. 4–25, 2016.

[9] K. Compton, B. Kybartas, and M. Mateas, "Tracery: an author-focused generative text tool," in *International Conference on Interactive Digital Storytelling*. Springer, 2015, pp. 154–161.

[10] L. Spector, "Assessment of problem modality by differential performance of Lexicase selection in genetic programming: A preliminary report," in *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*. Philadelphia, Pennsylvania, USA: ACM, 2012, pp. 401–408.

[11] I. Greenberg, *Processing: creative coding and computational art*. Apress, 2007.

[12] I. Bergstrom and R. B. Lotto, "Code bending: A new creative coding practice," *Leonardo*, vol. 48, no. 1, pp. 25–31, 2015.

[13] K. Peppler and Y. Kafai, "Creative coding: Programming for personal expression," vol. 30, no. 2008, p. 314, 2005.

[14] D. Shiffman, S. Fry, and Z. Marsh, *The nature of code*. D. Shiffman, 2012.

[15] N. Dehouche and K. Dehouche, "What is in a text-to-image prompt: The potential of stable diffusion in visual arts education," *arXiv preprint arXiv:2301.01902*, 2023.

[16] L. S. Vestergaard, J. Fernandes, and M. Presser, "Creative coding within the internet of things," in *2017 Global Internet of Things Summit (GIoTS)*. IEEE, 2017, pp. 1–6.

[17] J. Kari, "Theory of cellular automata: A survey," *Theoretical computer science*, vol. 334, no. 1-3, pp. 3–33, 2005.

[18] K. Perlin, "Noise hardware. in real-time shading," *SIGGRAPH Course Notes*, 2001.

[19] M. Olano, K. Akeley, J. C. Hart, W. Heidrich, M. McCool, J. L. Mitchell, and R. Rost, "Real-time shading," in *ACM SIGGRAPH 2004 Course Notes*, 2004, pp. 1–es.

[20] G. H. Weiss and R. J. Rubin, "Random walks: theory and selected applications," *Advances in Chemical Physics*, vol. 52, pp. 363–505, 1983.

[21] J. Doran and I. Parberry, "Controlled procedural terrain generation using software agents," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 2, pp. 111–119, 2010.

[22] W. Chau, S. Wong, and S. Wan, "A critical analysis of dithering algorithms for image processing," in *IEEE TENCON'90: 1990 IEEE Region 10 Conference on Computer and Communication Systems. Conference Proceedings*. IEEE, 1990, pp. 309–313.

[23] Lux. Dithering for pixel artists. [Online]. Available: https://pixelparmesan.com/dithering-for-pixel-artists/

[24] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," *Statistics and Computing*, vol. 4, no. 2, pp. 87–112, 1994.

[25] T. Helmuth, N. F. McPhee, and L. Spector, *Lexicase Selection for Program Synthesis: A Diversity Analysis*. Cham: Springer International Publishing, 2016, pp. 151–167.

[26] J. M. Moore and A. Stanton, "Lexicase selection outperforms previous strategies for incremental evolution of virtual creature controllers," in *Proceedings of the 14th European Conference on Artificial Life*. Lyon, France: MIT Press, 2017, pp. 290–297.

[27] J. Huizinga and J. Clune, "Evolving Multimodal Robot Behavior via Many Stepping Stones with the Combinatorial Multi-Objective Evolutionary Algorithm," *Evolutionary Computation*, pp. 1–34, 11 2021. [Online]. Available: https://doi.org/10.1162/evco_a_00301

[28] Y. He, C. Aranha, A. Hallam, and R. Chassagne, "Optimization of subsurface models with multiple criteria using lexicase selection," *Operations Research Perspectives*, vol. 9, p. 100237, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214716022000124

[29] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: solving problems with box constraints," *IEEE transactions on evolutionary computation*, vol. 18, no. 4, pp. 577–601, 2013.

[30] W. La Cava, L. Spector, and K. Danai, "Epsilon-lexicase selection for regression," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. Denver, Colorado, USA: ACM, 2016, pp. 741–748.

[31] J. M. Moore and A. Stanton, "Tiebreaks and diversity: Isolating effects in lexicase selection," in *Proceedings of the 16th International Conference on the Simulation and Synthesis of Living Systems*. Tokyo, Japan: ACM, 2018, pp. 590–597.

[32] J. Lehman and K. O. Stanley, "Novelty search and the problem with objectives," in *Genetic programming theory and practice IX*. Springer, 2011, pp. 37–56.

[33] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11. ACM, 2011, pp. 1–10.

[34] A. Stanton and J. M. Moore, "Lexicase selection for multi-task evolutionary robotics," *Artificial Life*, vol. 28, no. 4, pp. 479–498, 2022.

[35] K. Crowson, S. Biderman, D. Kornis, D. Stander, E. Hallahan, L. Castricato, and E. Raff, "Vqgan-clip: Open domain image generation and editing with natural language guidance," in *European Conference on Computer Vision*. Springer, 2022, pp. 88–105.

[36] J. Ploennigs and M. Berger, "Ai art in architecture," *arXiv preprint arXiv:2212.09399*, 2022.

[37] L. Floridi and M. Chiriatti, "Gpt-3: Its nature, scope, limits, and consequences," *Minds and Machines*, vol. 30, no. 4, pp. 681–694, 2020.