

Addressing the Fundamental Tension of PCGML with Discriminative Learning

Isaac Karth

University of California Santa Cruz
Department of Computational Media
ikarth@ucsc.edu

Adam M. Smith

University of California Santa Cruz
Department of Computational Media
amsmith@ucsc.edu

ABSTRACT

Procedural content generation via machine learning (PCGML) is typically framed as the task of fitting a generative model to full-scale examples of a desired content distribution. This approach presents a fundamental tension: the more design effort expended to produce detailed training examples for shaping a generator, the lower the return on investment from applying PCGML in the first place. In response, we propose the use of discriminative models, which capture the validity of a design rather than the distribution of the content, trained on positive and negative example design fragments. Through a modest modification of WaveFunctionCollapse, a commercially-adopted PCG approach that we characterize as using elementary machine learning, we demonstrate a new mode of control for learning-based generators. We demonstrate how an artist might craft a focused set of additional positive and negative design fragments by critique of the generator’s previous outputs. This interaction mode bridges PCGML with mixed-initiative design assistance tools by working with a machine to define a space of valid designs rather than just one new design.

CCS CONCEPTS

• **Computing methodologies** → *Machine learning approaches*; **Machine learning**; • **Applied computing** → *Media arts*; • **Human-centered computing** → *Interaction paradigms*.

KEYWORDS

machine learning, mixed-initiative interface, design tools, constraint solving, procedural content generation, pcgml, procedural content generation machine learning

ACM Reference Format:

Isaac Karth and Adam M. Smith. 2019. Addressing the Fundamental Tension of PCGML with Discriminative Learning. In *The Fourteenth International Conference on the Foundations of Digital Games (FDG '19)*, August 26–30, 2019, San Luis Obispo, CA, USA. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.1145/3337722.3341845>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

FDG '19, August 26–30, 2019, San Luis Obispo, CA, USA

© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-7217-6/19/08...\$15.00

<https://doi.org/10.1145/3337722.3341845>

1 INTRODUCTION

Procedural Content Generation via Machine Learning (PCGML) is the recent term for the strategy of controlling content generators using examples [33]. Existing PCGML approaches usually train their statistical models based on pre-existing artist-provided samples of the desired content. However, there is a fundamental tension here: machine learning often works better with more training data, but the effort to produce enough high-quality training data is frequently costly enough that the artists might be better off just making the final content themselves.

Rather than attempting to train a generative statistical model (capturing the distribution of desired content), we focus on applying *discriminative learning*. In discriminative learning, the model learns to judge whether a candidate content artifact would be valid or desirable independent of the process used to generate it. Pairing a discriminative model with a pre-existing content generator that can accept a validity constraint, we realize example-driven generation that can be influenced by both positive and negative example design fragments. We examine this idea inside of an already-commercially-adopted example-based generation system, WaveFunctionCollapse (WFC) [14].¹ This approach begins to address the fundamental tension in PCGML while also opening connections to mixed-initiative design tools, the source of the design fragments.

Mixed-initiative content generation tools [15] are usually designed around the idea of the artist having a conversation with the tool about one specific design across many alterations, with the goal of creating one high-quality design. We propose to adapt this *conversational teaching model* for application in PCGML systems. While still having conversations with the tool about specific designs, the conversations are leveraged to talk about the general shape of the design space (rather than one specific output [9]). The artist trains the overall generative system to the point where it will be trusted to follow that style in the future, when the system can be run non-interactively. Instead of an individual artifact, the goal is define a space of desirable artifacts from which the generator may sample.

This paper illuminates the implicit use of machine learning in WFC, explains how discriminative learning may be integrated, and presents a detailed worked example of the conversational teaching model. We refer to the primary user as an artist to emphasize the primarily visual interface. It should be understood that the process of creating the input image can involve both design skills and programming reasoning: the artist is specifying both an aesthetic goal and a complex system of constraints to achieve that goal.

¹<https://github.com/mxgmn/WaveFunctionCollapse>

2 BACKGROUND

In this section, we review WFC as an example-driven generator, characterize PCGML work to date as operating on only positive examples, and review the conversational interaction model used in mixed-initiative design tools.

2.1 WaveFunctionCollapse

WaveFunctionCollapse is a content generation algorithm devised by independent game developer Maxim Gumin. In contrast with generators presented in technical games research venues, WFC has seen surprisingly quick adoption within the technical artist community. Particularly notable is that WFC can be considered an instance of PCGML, as we illustrate in Sec. 3.

The Viking invasion game *Bad North* [31] uses Oskar Stålberg’s WFC implementation for generating island maps.² *Caves of Qud* [7], a roguelike that is currently in Early Access on Steam, uses WFC as one of its map generation techniques. The *Caves of Qud* developers have closely followed the ongoing development of the algorithm, incorporating its recent improvements.³ In particular, *Caves of Qud*’s implementation of the improved “fast WFC” enables it to use a higher N for its $N \times N$ patterns, which potentially allows the developers to express more complex structures.⁴

WFC is an instance of content generation using constraint solving techniques [14]. WFC analyzes an input image and expresses a weighted constraint satisfaction problem based on these local similarity properties. There are many alternative constraint solving systems that can be substituted for Gumin’s original observe-and-propagate cycle, including our own declarative implementations with the answer-set solver Clingo [6] and the recent “fast WFC” implementation by Mathieu Fehr and Nathanaël Courant [4].

In this paper, we seek to extend the ideas of WFC while keeping them compatible with the existing implementations. One of the more unique aspects of WFC is that it is an example-based generator that can generalize from a single, small example image. In Sec. 5 we show that while more than one example is needed to appropriately sculpt the design space, the additional examples can be even smaller than the original and can be created in response to generator behavior rather than collected in advance.

2.2 PCGML

Summerville et al. define Procedural Content Generation via Machine Learning (PCGML) as the “generation of game content by models that have been trained on *existing* game content [emphasis added]” [33]. In contrast with search-based and solver-based approaches which presume the user will provide an evaluation procedure or logical definition of appropriateness, PCGML uses a more artist-friendly framing that assumes concrete example artifacts as

the primary inputs. PCGML techniques may well apply constructive, search, or solver-based techniques internally after interpreting training examples.

Machine learning needs training data, and one significant source of data for PCGML research is the Video Game Level Corpus (VGLC), which is a public dataset of game levels [35]. The VGLC was assembled to provide corpora for level generation research, similar to the assembled corpora in other fields such as Natural Language Processing. In contrast with datasets of game level appearance such as VGMaps,⁵ content in the VGLC is annotated at a level suitable for constructing new, playable level designs (not just pictures of level designs).

The VGLC provides a valuable set of data sourced from iconic levels for culturally-impactful games (e.g. *Super Mario Bros* and *The Legend of Zelda*). It has been used for PCG research using autoencoders [13], generative adversarial networks (GANs) [36], long short-term memories (LSTMs) [32], multi-dimensional Markov chains [27, Sec. 3.3.1], and automated game design learning [20].

Some attempted solutions involve leveraging existing data. Snodgrass and Ontañón [28] train generative models but address the problem of small training data via transfer learning: training on data from related domains (level designs for other videogames). Sarkar and Cooper [24] similarly use data from adjacent domains to create novel designs via blending. However, Summerville et al. identify a “recurring problem of small datasets” [33]: most data only applies to a single game, and even with the efforts of the VGLC the amount of data available is small, particularly when compared to the more wildly successful machine learning projects. This is further complicated by our desire to produce useful content for novel games (for which no pre-existing data is available). Hence the fundamental tension in PCGML: asking an artist (or a team of artists) to produce quality training data at machine-learning scale could be much less efficient than just having the artists make the required content themselves.

Compounding this problem, a study by Snodgrass et al. [30] showed that the expressive volume of current PCGML systems did not expand much as the amount of training data increased. This suggests that the generative learning approach taken by these systems may not ever provide the required level of artist control. While this situation might be relieved by using higher-capacity models, the problem of the effort to produce the training data remains.

PCGML should be compared with other forms of example-based generation. Example-based generation long predates the recent deep learning approaches, particularly for texture synthesis. To take one early example, David Garber’s 1981 dissertation proposed a two-dimensional, Markov chain, pixel-by-pixel texture synthesis approach [5]. Separate from Garber,⁶ Alexei Efros and Thomas Leung contributed a two-dimensional, Markov-chain inspired synthesis method: as the synthesized image is placed pixel-by-pixel, the algorithm samples from similar local windows in the sample image and randomly chooses one, using the window’s center pixel as the new value [3]. Although WFC-inventor Gumin experimented with continuous color-space techniques descending from these traditions, his use of discrete texture synthesis in WFC is directly

²As discussed in e.g. @OskSta: “The generation algorithm is a spinoff of the Wave Function Collapse algorithm. It’s quite content agnostic. I have a bunch of tweets about it if you scroll down my media tweets” <https://twitter.com/OskSta/status/931247511053979648>

³@unormal: “Got Qud’s new 20x faster WFC implementation down to about 50mb in-memory static overhead on init with 0 allocation for all future runs (unless the size of the gen output gets bigger, but Qud’s doesn’t), so no GC churn. (cc @ExUtmno)” <https://twitter.com/unormal/status/984713110257852416>

⁴@unormal, 2:05 AM - 13 Apr 2018: “20x faster also makes higher orders of N practical, which enables larger scale structures to pop out of wfc.” <https://twitter.com/unormal/status/984719207156862976>

⁵<http://vgmaps.com/>

⁶Efros and Leung later discovered Garber’s previous work [2].

inspired by Paul Merrell’s discrete model synthesis and Paul Harrison’s declarative texture synthesis [8]. Harrison’s declarative texture synthesis exchanges the step-by-step procedure used in earlier texture synthesis methods for a declarative texture synthesis approach, patterned after declarative programming languages [11, Chap. 7]. Merrell’s discrete 3D geometric model synthesis uses a catalog of possible assignments and expresses the synthesis as a constraint satisfaction problem [18]. Unlike later PCGML work, these example-based generation approaches only need a small number of examples, often just one. However, each of these approaches use only positive examples without any negative examples.

The strategy of avoiding directly learning a generative model explored in this paper is similar to the conceptual move in Generative Adversarial Networks (GANs). Rather than a training a directly generative model, GANs co-train distinct generator and discriminator models. GANs have been used for PCGML, for example in MarioGAN [37], which uses unsupervised learning trained on levels from the Video Game Level Corpus (VGLC) [34] to generate levels based on the structure of *Super Mario Bros*.

Our work also uses example data to influence a distinct generator and discriminator, but these models are not represented by differentiable functions: they are instead represented by systems of constraints. Through a generate-and-test approach, learned validity constraints can be layered onto existing generative methods. For WFC, in particular, the validity constraints can be integrated into the core algorithm itself (a form of constraint solving).

Related to our approach, Guzdial et al. consider a use for discriminative learning in PCGML [10]. Rather than training a pattern-validity classifier (as in our work), they train a design pattern label classifier with labels such as "intro" and "staircase". While these labels do not reshape the space of (Mario level) designs the system will output, they allow the generator to annotate its outputs in designer-relevant terms.

2.3 Mixed-Initiative Design Tools

Several mixed-initiative design tools have integrated PCG systems. Their interaction pattern can be generalized as an iterative cycle where the generator produces a design and the artist responds by making a choice that contradicts the generator’s last output. When the details of a design are under-constrained, most mixed-initiative design tools will allow the artist to re-sample alternative completions.

Tanagra is a platformer level design tool that uses reactive planning and constraint solving to ensure playability while providing rapid feedback to facilitate artist iteration [26]. Additionally, Tanagra maintains a higher-order understanding of the beats that shape the level’s pacing, allowing the artist to directly specify the pacing and see the indirect result on the shape of the level being built.

The SketchaWorld modeling tool introduces a declarative “procedural sketching” approach “in order to enable designers of virtual worlds to concentrate on stating what they want to create, instead of describing how they should model it” [25]. The artist using SketchaWorld focuses on sketching high-level constructs with instant feedback about the effect the changes have on the virtual world being constructed. At the end of interaction, just one highly-detailed world results.

Similarly, artists interact with Sentient Sketchbook via map sketches, with the generator’s results evaluated by metrics such as playability. As part of the interactive conversation with the artist, it also presents evolved map suggestions to the user, generated via novelty search [16, 17]. Although novelty search could be used to generate variations on the artist’s favored design, it is not assumed that all of these variations would be considered safe for use. Tools based on interactive evolution do not learn a reusable content validity function nor do they allow the artist to credit or blame a specific sub-structure of a content sample as the source of their fitness feedback. In Sec. 4, we demonstrate an interactive system that can do both of these.

As these examples demonstrate, mixed-initiative tools facilitate an interaction pattern where the artist sees a complete design produced by a generator and responds by providing feedback on it, which influences the next system-provided design. This two-way conversation enables the artist to make complex decisions about the desired outcome without requiring them to directly master the technical domain knowledge that drives the implementation of the generator. The above examples demonstrate the promising potential of design tools that embrace this mode of control. However, they tend to focus on using generation to assist in creating specific, individual artifacts rather than the PCGML approach of modeling a design space or a statistical distribution over the space of possible content.

Despite the natural relationship between artist-supplied content and the ability of machine learning techniques to reflect on that content and expand it, PCGML-style generators that learn during mixed-initiative interaction have not been explored beyond the recent Morai-Maker-Engine [9]. In 2019, this discussion can be framed in terms of *human-centered machine learning* [23].⁷

3 CHARACTERIZING WFC AS PCGML

Snodgrass describes WaveFunctionCollapse as “an example of a machine learning-based PCG approach that does not require a deep understanding of how the algorithm functions in order to be used effectively” [27, Sec. 2.8]. This section describes what and how WFC learns in a generalized vocabulary, which we introduce in this paper, that opens up space for exploring alternative learning strategies.

Rather than being a single monolithic input-to-output-mapping function, WaveFunctionCollapse is a pipeline of multiple stages. While most prior discussion has emphasized the constraint-solving stage, the pattern learning steps that precede it are equally important. It starts with an analysis of an image to a vocabulary of valid adjacencies between patterns. In the second phase, the results of the analysis are used as constraints in a generation process identifiable as constraint solving [14].

In this section, we describe how the pattern learning steps are an instance of machine learning. In particular, we show that this phase learns three functions: which pattern is present at each location (the pattern classifier); if the pattern adjacency pairing is within the artist’s preferred style (the adjacency validity); and how a location in the output should be rendered (the pattern renderer) given the the constraint-solving generator’s choice of pattern placement

⁷Such as the focus of the Human-Centered Machine Learning Perspectives Workshop <https://gonzoramos.github.io/hcmllperspectives/>

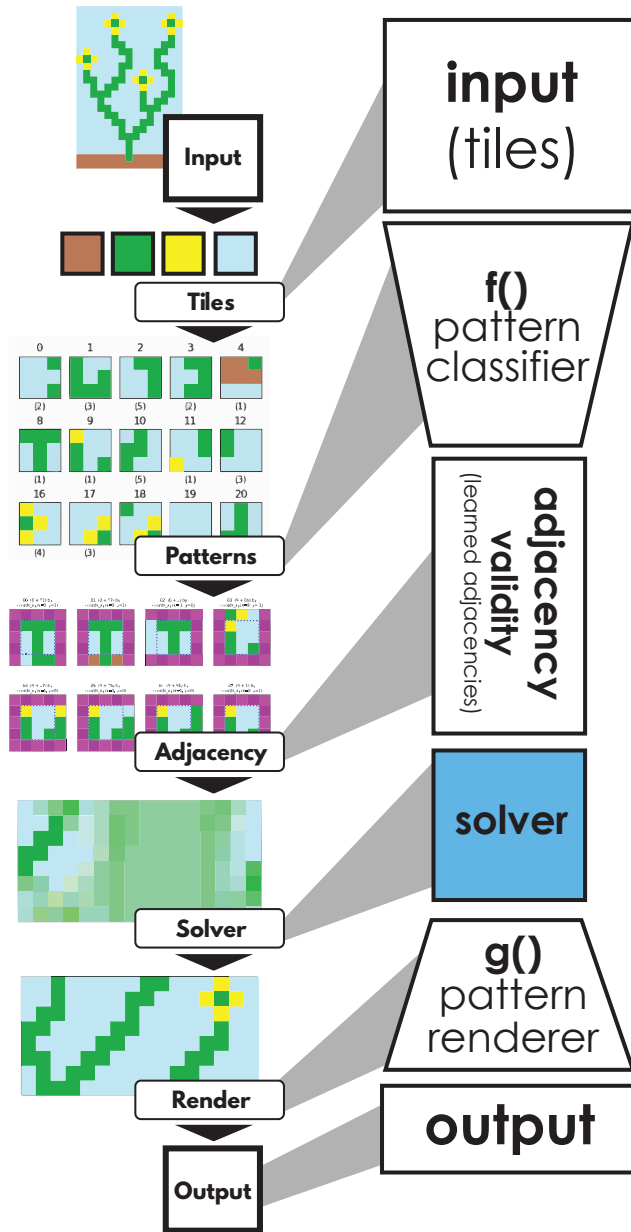


Figure 1: WaveFunctionCollapse pipeline. The left column shows the pipeline steps. The right column indicates how the steps match the functions learned from pattern analysis.

assignments. Fig. 1 illustrates the relationship between these three functions.

3.1 Tiles

Conventionally, the space WFC operates on is a rectangular grid. This gives us an easy way to define the nodes and adjacency edges: the nodes are each intersection on the Go board, and the edges are the lines between them. However, WFC can work on any graph for which a well-defined adjacency function can be specified. For

example, Stålberg’s experiments with triangular meshes.⁸ These can be non-spatial: to create a looping animation Matt Rix added a time edge.⁹ Similarly, Martin O’Leary’s poetry generator¹⁰ uses non-neighbor edges for rhyming patterns and scansion.

Multi-node tiles are also possible: one of the *Bad North* [31] innovations was to include larger elements as possible modules.¹¹

WaveFunctionCollapse works by placing small elements into the context of a larger whole. While these puzzle pieces go by many different names in the wild we refer to them as tiles. A tile can be a single pixel, a 2D image, 3D geometry, a word, or any other distinct modular component.

Typically, tiles are either specified by an artist or extracted from the training data. The solver does not care about the contents of the tiles, only about the adjacencies between tiles.

3.2 Pattern Classifier

The heart of WFC are the *adjacencies* that describe the constraints between patterns, used by the constraint solver to generate the solution. Gumin’s original implementation of WFC has ways of defining patterns: a SimpleTileModel that specifies adjacencies between individual tiles by hand and an OverlappingModel that infers the relationships between tiles in their local contexts. Other models are possible: for example, *Bad North* [31] learns which modules can be adjacent by comparing the vertices at the edges of the 3D model and looking for matching profiles in the shared plane.

With this learning step, the OverlappingModel operates on what it refers to as *patterns*. A pattern is a tile plus the context of its surrounding adjacencies, as found in the training data. These are usually $N \times N$ regions of tiles (where N is typically 2 or 3, as larger values need more training data and computing resources). The classifier reduces pertinent details about the surrounding context into a single value. By operating on patterns rather than directly on tiles, the solver can make use of the implied higher-order relationships that are learned from the training data.

The OverlappingModel also uses reflection and symmetry to augment the training data. Readers with a machine learning background will recognize this as a *data augmentation* strategy [1, p. 138-142]. This can be configured for the training since some input images, such as a flower viewed from the side, have a strong directionality.

In Gumin’s implementation of WaveFunctionCollapse, the pattern classifier is a relatively simple 1:1 mapping of patterns learned from the training data. Other learning methods are possible: for example, grouping semantically similar tiles via k-means. Or we can imagine a deep convolutional neural network being used to map as-yet unseen tile configurations into the existing pattern catalog so long as they were perceptually similar enough.

Pattern classification need not be a strictly local operation. If we wanted to generate dungeon levels for a roguelike game, we may be particularly interested to note which treasure chests are easily reachable by the player versus not. Or in platformer level generation

⁸<https://twitter.com/OskSta/status/784847588893814785>

⁹<https://twitter.com/MattRix/status/872674537799913472>

¹⁰<https://github.com/mewo2/oisin>

¹¹ A conceptually simpler way to implement multi-tile elements is to give the different parts the constraint that the only allowed neighbor in the relevant edge is another part of the multi-tile module.

this could distinguish rewards placed on the player’s default path (as a guide) or off the path (as an enticement to explore). In the future, we imagine the contextual information used in the pattern classifier to come from many different sources. In the *texture-by-numbers* application of image analogies [12], the artist hand-paints an additional input image to guide the interpretation of the source image and the generation of the target image in another example-driven image generator. In Snodgrass’ hierarchical approach to tile-based map generation [29], a lower-resolution context map is generated automatically using clustering of tile patterns.

3.3 Patterns and Adjacency

Gumin’s WFC operates not on the tile constraints, but rather on the constraints between patterns (Fig. 2). This is a generalization of the adjacencies between individual tiles: the pattern classifier captures additional adjacency information about the local space, similar to an image filter kernel or the convolutions used in image processing and CNNs. In effect, each tile is treated as the tile-plus-its-context: we prefer some adjacencies, such as placing a flower in a flower-pot. Other adjacencies are non-preferred: the flower should not be growing in the middle of a carpet. In Gumin’s WFC implementation adjacencies are stored as a (usually sparse) multidimensional matrix, with dimensions of $patterns \times patterns \times directions$.¹²

We can characterize the method used to learn the adjacency legality as Most General Generalization (MGG), the inverse of classic Least General Generalization (LGG) inductive inference technique [21]. Gumin’s implementation simply allows any tile-compatible overlapping patterns to be placed adjacent to one another, even if they were never seen adjacent in the single source image. A side effect of this is that any pattern adjacencies seen in the source image (which are tile-compatible by construction) must be considered valid for the generator to use later. While MGG might appear as simple parsing and tallying, something too simple to be considered as machine learning, it is useful to compare this approach with other classic machine learning techniques like Naive Bayes [22, Chap. 20]. Naive Bayes classifiers are trained with no more sophistication than tallying how often each feature was associated with each class.

3.4 Using Multiple Sources of Training Data

The art of constructing the single source image for Gumin’s WFC often involves some careful design to include all of the patterns that are preferred and none that are non-preferred. By allowing for multiple positive and negative examples and using a slightly altered learning strategy, we show how this meticulous work can be replaced with a conversation that elaborates on past examples.

While many WFC implementations use a single image for training data, this is an interface detail rather than an intrinsic limitation of the algorithm. Using multiple images allows for discontinuities in the training data, which simplifies the expression of some complex relationships. Equally, since the adjacencies between patterns are just a set of tuples¹³ we can also use *negative examples* (at the cost

¹²Constraint solvers that are not optimized for grids of constraints can, instead, use a set of the allowed adjacencies to specify the constraints. This list of tuples is isomorphic with the pattern matrix and is more convenient for adding or subtracting adjacencies from the allowed set.

¹³ or a sparse matrix

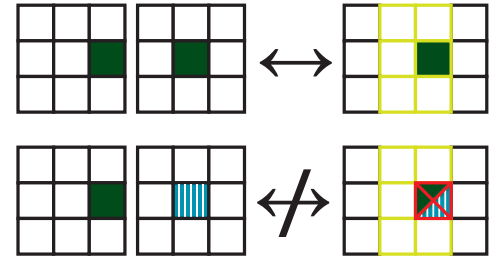


Figure 2: An example of a pattern overlap, with a [0,1] offset. The top pair of patterns is a legal overlap, because of the valid intersection. The bottom pair is not a legal overlap, because the striped blue tile and the solid green tile conflict.

of increased interface complexity) that remove adjacencies from the allowed set.

One of the reasons that WFC was rapidly adopted was that artists could create complex constraints by painting a picture. Complicating the interface removes some of this advantage. However, other equally approachable ways to specify constraints have already been explored—for example, *Bad North* [31] automatically detects alignment between the 3D geometry of neighboring tiles.

Gumin’s pattern classifier function implicitly captures the relationships between patterns in the training data. The first and most absolute distinction is between legal and illegal overlaps: because the patterns in the *OverlappingModel* need to be able to be placed on top of each other without contradictions, some patterns will never be legal neighbors: if one 3×3 pattern has a blue center tile, while another 3×3 pattern has a green right tile, the green-right-tile pattern can never be legally placed to the left of the blue-center-tile pattern (Fig. 2).

Gumin’s MGG learning strategy is hardly the only option possible even using the default pattern classifier. A LGG learning strategy would say to only allow those adjacencies explicitly demonstrated in the source image. However, this highly-constrained alternative might not allow any new output to be constructed that was not an exact copy of the source image. Likely, the ideal amount of generalization falls somewhere between these extremes.

In a discriminative learning setup, we might consider all adjacencies explicitly demonstrated in positive example images to be therefore positive examples for the learned adjacency relation. Likewise, a negative example image needs to demonstrate at least one adjacency that would be considered invalid by the learned relation. We refer to the artist’s intended set of allowed relations the *preferred set*. Artists can attempt to adjust the legal set to match the preferred set, but this requires a combination of technical reasoning and trial-and-error iteration.

This is a limitation of the learning strategy, but not the WFC algorithm itself. The output of the `agrees()` validity function in Gumin’s implementation just checks if two patterns can legally overlap, but any arbitrary adjacency validity function can be substituted here. As long as the validity function can be computed over all pairs of patterns, it can act as the whitelist for the constraint domains without changing the WFC solver itself.

3.5 Additional Pre-Solving Constraints

While most uses of WFC to date have encoded all information about the long-range constraints in the training image data, an implementation with a more specific application in mind has the opportunity to include additional constraints. For example, including a reachability constraint in a level generator (so that there is a path to all the rooms in the level) can be implemented as a global constraint.

The most common use of additional constraints in the wild is to pre-seed the solver with partial solutions. For example, the *Flowers* example is pre-seeded with the lowest row limited to patterns that include the brown soil pixels. Similarly, the *Caves of Qud* level generator uses WFC as part of a pipeline, with additional details added in the empty spaces between the walls that WFC adds. One useful side-effect of this is that WFC can complete partial solutions. When used as part of a mixed-initiative generator, this means that the user can draw a partial solution (such as the main path through a level) and have the generator fill in the rest of the space with relevant and contextual content.

3.6 Solver

The adjacency data is sent to the constraint solver. Only the constraint data itself is needed: the list of constraints is sufficient. The constraint solver can be implemented as Gumin’s stochastic observation/propagation solver; an ASP solver like Clingo¹⁴; a solver that uses a constraint modeling language, such as MiniZinc¹⁵ and so on. Though it is not the focus of the present paper, the properties of the Solver can vary. Most commonly this takes the form of different heuristics or the addition of backtracking, which is most applicable for tile sets that greatly depart from the properties of the original examples (small N, adjacencies that are neither too constrained or too open). With Gumin’s parameters, the long-range constraint propagation in WFC is more important for its high success rate without backtracking, rather than the heuristic used [14].

3.7 Rendering

The final step of WFC is an inversion of the pattern classification: translating the grid of selected patterns back into a grid of tiles (and tiles into pixels). Interesting animations showing the progress of generation in WFC result from blending the results of the pattern renderer for all patterns that might yet still be placed at a location. Animations of these visualizations over time attracted several technical artists (and the present authors) to learn more about WFC.

Generalizing the role of the pattern classifier, we can imagine other functions which decide how to represent a local patch of pattern placements. Again, we can imagine the use of a deep convolutional neural network (CNN) to map a small grid of pattern identifier integers into an rich display in the output. Although the pattern renderer’s input datatype is fixed, the output can be whatever artist-visible datatype was used as input to the pattern classifier (whether that be image pixels, game object identifiers, or a parameter vector for a downstream content generator).

If additional annotation layers are used in the source images (as in the texture-by-numbers application mentioned above, the

navigability criteria in *Bad North*,¹⁶ or the player path data present in some VGLC data), it is reasonable to expect that the output of the generator could also have these annotation layers. For platformer level generation, the system could output not only a tile-based map design, but also a representation of which parts of the map player can actually reach. Likewise, if the tiles are more complex than static images, the rendering function can output things other than the final image. For example, a level generator might have each tile represent a *room generator* rather than the final level geometry, and the rendering output would be the parameter data for the calls to the room generators.

3.8 Variation in Implementations

Each of the above steps can be replaced. The many implementations of WFC in the wild frequently vary the features that each step uses: for example, *Bad North* doesn’t use patterns to learn tile adjacency, instead relying on matching geometric profiles of 3D models.

To prototype a variation of WFC supporting an alternate pattern classifier, pattern renderer, and adjacency validity function, we initially developed a surrogate implementation of WFC in the MiniZinc constraint programming language. Later, we integrated the specific ability to generate with a customized pattern adjacency whitelist into a direct Python-language clone of Gumin’s original WFC algorithm.

4 SETTING UP DISCRIMINATIVE LEARNING

As discussed above, the original approach in WFC is to define the adjacency validity function with the most permissive possible way, using the every possible legal adjacency as the valid set. Among other drawbacks, this requires careful curation of the patterns so that every adjacency in the legal set is acceptable. While allowing very expressive results from a single very small source image, there are many preferred sets that are difficult to express in this manner. However, this is just one of the many possible strategies. An anomaly-detection strategy, such as a one-class Support Vector Machine [19], might allow the set of valid adjacencies to more closely approximate the ideal preferred set, allowing the artist to use patterns with a much larger legal adjacency set.

In this section, we consider the presence of possible negative examples. By removing adjacency pairs that the artist explicitly flagged as undesirable, we can more precisely determine the valid set. By default, WFC uses all of the legal adjacencies, but the preferred-valid set can be adjusted to include more or less of the legal set, with corresponding effects for the generation.

4.1 Machine Learning Setup

In addition to the single positive source image used by the original WFC, we introduce the possibility of using more source images. Some of these are additional positive examples: it can be easier to express new adjacencies while avoiding unwanted ones by adding a completely separate positive example. In the positive examples, every adjacency is considered to be valid, as usual. In contrast, sometimes it is easier to specify just the negative adjacencies to be removed from the preferred set. Note that these additional example images do not need to be equal in size. In fact, a tiny negative

¹⁴<https://github.com/potassco/clingo>

¹⁵<https://www.minizinc.org/>

¹⁶<https://twitter.com/OskSta/status/917405214638006273>

example showing just the undesirable pairing lets an artist carve out one bad location in an otherwise satisfactory design.

Finally, we have the validity function: a function that takes two patterns (plus how they are related in space, e.g. up/down/left/right) and outputs a Boolean evaluation of whether or not their adjacency is valid. In the original WFC, this is simply an overlapping test: given this offset, are there any conflicts in the intersection of the two patterns? However, as suggested above, there are more sophisticated validity functions that also produce viable results.

4.2 Human Artist Setup

In our mixed-initiative training approach, we expect the artist to provide at least one positive example to start the process. The image should demonstrate the local tiles that might be used by the generator, but it does not need to demonstrate all preferred-valid adjacencies. Note that providing one example is the typical workflow for WFC (unmodified, Gumin’s code only accepts one example). However, instead of expecting the artist to continue to iterate by changing this one example, which can quickly grow complex, the artist can isolate each individual contribution.

Initially, we set the whitelist of valid adjacencies to be fully permissive (MGG), covering the legal adjacencies of the known patterns. From this, we generate a small portfolio of outputs to sample the current design space of the generator. Even a single work sample is often enough to spur the next round of interaction. The artist reviews the portfolio to find problems. They can add one or more generated outputs to the negative example set directly, crop an example to make a more focused negative example, or hand-create a clarifying example. If additional positive examples are desired to increase variety, those can also be added (although they may immediately prompt the need for negative examples to address over-generalization).

With the new batch of source images, in a trivial amount of time we retrain each of the learned functions: the pattern classifier, the adjacency validity function, and the pattern renderer. The update pattern classifier defines the space of patterns that might be placed by the generator. The newly-learned validity function defines the updated whitelist used in the constraints. The updated pattern renderer might even display existing pattern grids in a new way. As before, we sample a portfolio. The artist repeats the process until they are satisfied with the work samples. The result is a generative system with a design space that has been sculpted to the artist’s requirements, all without the artist needing to understand or alter any unfamiliar machine learning algorithms.

5 WORKED EXAMPLE

In this section we walk through an example run of the conversational interaction an artist has with WFC when using a discriminative learning setup. The conversation takes place over several iterations that are visually represented in Fig. 3. All outputs shown were generated by executing a minimally modified WFC implementation. The only alterations are adding patterns from multiple images paired with removing items from the adjacency whitelist. The resulting pattern grids are rendered with the pattern renderer.

In this running example, we make use of a refinement to the MGG strategy used in Gumin’s implementation. Rather than simply

allowing all patterns which agree on their overlapping tiles, we allow all such patterns *except* those taken from negative examples. Indeed, this is still the most general generalization possible under the extra constraints. Working through the conversation, our artist begins Iteration 1 with the algorithm by supplying a single positive example. Here we use the Flowers example taken from Gumin’s public repository. MGG learns the legality relations exactly like the original WFC. Using the generator to produce work sample, the artist decides that the image needs more colorful flowers.

In Iteration 2, the artist augments the positive image set with a second image, having repainted the flowers to be red. Adding additional patterns to Gumin’s WFC required only minimal code changes. In the resulting work sample, now both red and yellow flowers are seen (new patterns were made available to the generator). However, the artist is still wants more flower variety.

Rather than copy-pasting the original tiles again, this time the artist creates a number of smaller samples that focus exactly on what they want to add to the composition. These extra tiny examples might throw off statistics in a generatively trained model. In the work sample for Iteration 3, the new flowers are present but a surprising new phenomenon arises. This possibility of floating stems results from the particulars of WFC’s default pattern classifier and adjacency relation function learning. The artist is not concerned with these implementation details and wishes simply to fix the problem with additional examples.

By selecting and cropping a 3×4 region of the last work sample, the artist creates the focused negative example for use in Iteration 4. MGG, now with the extra constraint from the negative example, no longer considers floating stems to be a possibility despite the fact that this pattern can be identified in the input by the pattern classifier. The work sample is now free from obvious flaws.

Having previously only considered very small training examples, the artist notes a particular feature of the larger generated output. The ground appears uninterestingly flat. In Iteration 5, the artist provides a small positive example of sloped hills, hoping the generator will invent a rolling landscape. However, the work sample for this iteration suggests that the generator has not picked up the generality of the idea from the single tiny example—it only knows how to build continuous ramps without any flowers.

In Iteration 6, a few more positive examples show that stems can be placed on hills and that the bumps of hills can be isolated (not always part of larger ramps). However, in rare circumstances the generator will now place stems underground. This would not have been spotted without examining many possible outputs, highlighting the importance of a tool that allows the artist to give feedback on more than one example output.

Finally, in Iteration 7, the artist is able to get the look they preferred. Adding negative examples to take care of the edge cases is easy and can be done without adjusting the earlier source images. Testing shows that the generator is reliably producing usable images. Because of the iterations, the artist now has enough trust in the generator to allow it to perform future generation tasks without supervision. The learned pattern classifier function, pattern renderer function, and adjacency legality function compactly summarize the learning from the interaction with the artist.

In this worked example, every new training example added beyond the first is a direct response to something observed in (or


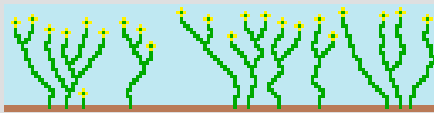
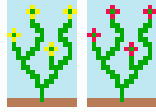
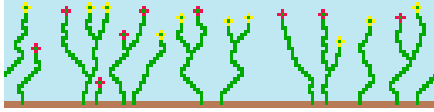
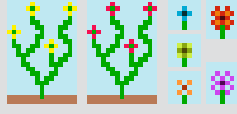
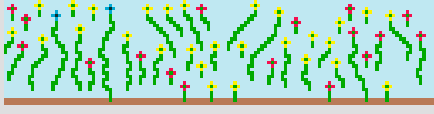
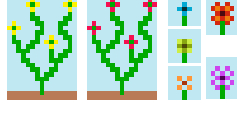

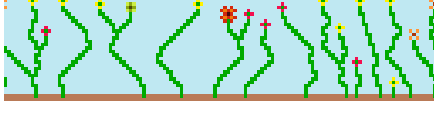
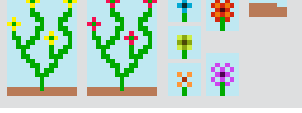

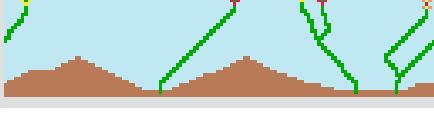


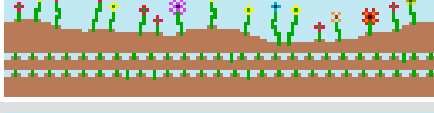
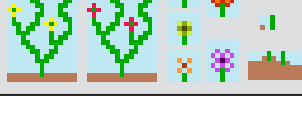

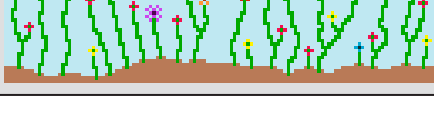
	Positive	Negative	One Output Sample	
1				Original, single example image: only generates yellow flowers Artist wants more variety, adds red flowers
2				Now it generates both red and yellow flowers Artist wants more variety, adds more flowers, but only blossoms
3				Stems aren't anchored, flowers are floating in the sky Artist adds a negative example to forbid floating stems
4				Many varieties of flowers bloom. Artist thinks it looks too flat, adds hills.
5				No longer flat, but flowers aren't growing on top of the hills Artist replaces hill image with more targeted hill images.
6				A rare side-effect causes underground stems to grow Artist adds negative examples, forbidding those adjacencies
7				Flowers now grow on top of gentle rolling hills. Artist trusts generator, will now allow it to act autonomously

Figure 3: A worked example of the mixed-initiative conversational teaching model process. The artist observes the results of each step (top black text) and makes a change for the next step (lower blue text). Each step adds either positive or negative examples. The source images for each output can be seen in the columns on the left, with a representative output image to the right.

observed to be missing from) concrete images produced by the previous generator. Many demonstrate patterns that are not what the generator should produce in the future, even if it is not realized that this was the case earlier. Instead of iterating to produce a carefully curated set of 100% valid examples, we make progress by adding focused clarifications.

6 CONCLUSION

The fundamental tension in PCGML is that the effort to craft enough training data for effective machine learning might undermine the motivation to use PCGML in the first place. This makes many machine learning approaches impractical: even when the design goal is flexibility (rather than nominally infinite content) the immense amount of training data required can be daunting.

However, existing approaches to single-example PCG such as WaveFunctionCollapse suggests that small-training-data generators

are possible. When we combine them with a discriminative learning strategy, we can leverage the usefulness of focused negative examples, even just example fragments. As our worked example of the conversational teaching model shows, an artist can intuitively make targeted changes without being overly concerned about maintaining a representative distribution or disturbing earlier, carefully planned patterns just to fix a rare edge case.

Combining PCGML with mixed-initiative design assistance tools can enable artists to sculpt a generator's design space. Rather than building just one high-quality artifact, the artist can train a generator through iterative steps to the point where they trust it for autonomous generation.

REFERENCES

- [1] Francois Chollet. 2017. *Deep Learning with Python* (1st ed.). Manning Publications Co., Greenwich, CT, USA.
- [2] Alexei A. Efros and William T. Freeman. 2001. Image Quilting for Texture Synthesis and Transfer. In *Proceedings of the 28th Annual Conference on Computer*

- Graphics and Interactive Techniques (SIGGRAPH '01)*. ACM, New York, NY, USA, 341–346. DOI : <http://dx.doi.org/10.1145/383259.383296>
- [3] Alexei A Efros and Thomas K Leung. 1999. Texture synthesis by non-parametric sampling. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, Vol. 2. IEEE, IEEE Computer Society, 1999, 1033–1038.
 - [4] Mathieu Fehr and Nathanaël Courant. 2018. fast-wfc. <https://github.com/mathfehr/fast-wfc>, *GitHub repository* (2018).
 - [5] David Donovan Garber. 1981. *Computational Models for Texture Analysis and Texture Synthesis*. Ph.D. Dissertation. University of Southern California, Los Angeles, CA, USA. AAI0551115.
 - [6] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. 2014. Clingo = ASP + Control: Preliminary Report. *CoRR* abs/1405.3694 (2014).
 - [7] Jason Grinblat and Charles B. Bucklew. 2010. Caves of Qud. (2010).
 - [8] Maxim Gumin. 2017. WaveFunctionCollapse Readme.md. (18 May 2017). Retrieved May 20, 2017 from <https://github.com/mxgmn/WaveFunctionCollapse/blob/master/README.md>
 - [9] Matthew Guzdial, Nicholas Liao, and Mark Riedl. 2018. Co-Creative Level Design via Machine Learning. In *The 5th Experimental AI in Games Workshop (EXAG)*. http://ceur-ws.org/Vol-2282/EXAG_126.pdf
 - [10] Matthew Guzdial, Joshua Reno, Jonathan Chen, Gillian Smith, and Mark Riedl. 2018. Explainable PCGML via Game Design Patterns. In *The 5th Experimental AI in Games Workshop (EXAG)*. http://ceur-ws.org/Vol-2282/EXAG_107.pdf
 - [11] Paul Francis Harrison. 2006. *Image Texture Tools: Texture Synthesis, Texture Transfer, and Plausible Restoration*. Monash University. 95–117 pages.
 - [12] Aaron Hertzmann, Charles E Jacobs, Nuria Oliver, Brian Curless, and David H Salesin. 2001. Image analogies. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 327–340.
 - [13] Rishabh Jain, Aaron Isaksen, Christoffer Holmgård, and Julian Togelius. 2016. Autoencoders for level generation, repair, and recognition. In *Proceedings of the ICCG Workshop on Computational Creativity and Games*.
 - [14] Isaac Karth and Adam M. Smith. 2017. WaveFunctionCollapse is Constraint Solving in the Wild. In *Proceedings of the 12th International Conference on the Foundations of Digital Games (FDG '17)*. ACM, New York, NY, USA, Article 68, 10 pages. DOI : <http://dx.doi.org/10.1145/3102071.3110566>
 - [15] Antonios Liapis, Gillian Smith, and Noor Shaker. 2016. Mixed-initiative content creation. In *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*, Noor Shaker, Julian Togelius, and Mark J. Nelson (Eds.). Springer, 195–216.
 - [16] Antonios Liapis, Georgios N Yannakakis, and Julian Togelius. 2013. Sentient Sketchbook: Computer-aided game level authoring. In *FDG*. 213–220.
 - [17] A. Liapis, G. N. Yannakakis, and J. Togelius. 2014. Designer modeling for Sentient Sketchbook. In *2014 IEEE Conference on Computational Intelligence and Games*. 1–8. DOI : <http://dx.doi.org/10.1109/CIG.2014.6932873>
 - [18] P. Merrell and D. Manocha. 2011. Model Synthesis: A General Procedural Modeling Algorithm. *IEEE Transactions on Visualization and Computer Graphics* 17, 6 (June 2011), 715–728. DOI : <http://dx.doi.org/10.1109/TVCG.2010.112>
 - [19] K. R. Müller, S. Mika, G. Ratsch, K. Tsuda, and B. Scholkopf. 2001. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks* 12, 2 (Mar 2001), 181–201. DOI : <http://dx.doi.org/10.1109/72.914517>
 - [20] Joseph Osborn, Adam Summerville, and Michael Mateas. 2017. Automatic Mapping of NES Games with Mappy. In *Proceedings of the 12th International Conference on the Foundations of Digital Games (FDG '17)*. ACM, New York, NY, USA, Article 78, 9 pages. DOI : <http://dx.doi.org/10.1145/3102071.3110576>
 - [21] Gordon D Plotkin. 1971. A further note on inductive generalization. *Machine intelligence* 6, 101-124 (1971).
 - [22] Stuart J. Russell and Peter Norvig. 2016. *Artificial Intelligence: A Modern Approach* (3 ed.). Pearson Education.
 - [23] Dominik Sacha, Michael Sedlmair, Leishi Zhang, John A. Lee, Jaakko Peltonen, Daniel Weiskopf, Stephen C. North, and Daniel A. Keim. 2017. What you see is what you can change: Human-centered machine learning by interactive visualization. *Neurocomputing* 268 (2017), 164 – 175. DOI : <http://dx.doi.org/10.1016/j.neucom.2017.01.105> Advances in artificial neural networks, machine learning and computational intelligence.
 - [24] Anurag Sarkar and Seth Cooper. 2018. Blending Levels from Different Games using LSTMs. In *The 5th Experimental AI in Games Workshop (EXAG)*. http://ceur-ws.org/Vol-2282/EXAG_125.pdf
 - [25] R. M. Smelik, T. Tutenel, K. J. De Kraker, and R. Bidarra. 2011. Semantic 3D Media and Content: A Declarative Approach to Procedural Modeling of Virtual Worlds. *Comput. Graph.* 35, 2 (April 2011), 352–363. DOI : <http://dx.doi.org/10.1016/j.cag.2010.11.011>
 - [26] Gillian Smith, Jim Whitehead, and Michael Mateas. 2011. Tanagra: Reactive planning and constraint solving for mixed-initiative level design. *IEEE Transactions on Computational Intelligence and AI in Games* 3, 3 (2011), 201–215.
 - [27] Sam Snodgrass. 2018. *Markov Models for Procedural Content Generation*. Ph.D. Dissertation. Drexel University.
 - [28] Sam Snodgrass and Santiago Ontanon. 2016. An Approach to Domain Transfer in Procedural Content Generation of Two-Dimensional Videogame Levels. (2016). <https://aaai.org/ocs/index.php/AIIDE/AIIDE16/paper/view/13985>
 - [29] Sam Snodgrass and Santiago Ontanon. 2015. A hierarchical MDMC approach to 2D video game map generation. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*.
 - [30] Sam Snodgrass, Adam Summerville, and Santiago Ontañón. 2017. Studying the Effects of Training Data on Machine Learning-Based Procedural Content Generation. In *Proceedings of the Thirteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-17)*. 122–128.
 - [31] Oskar Stålberg, Richard Meredith, and Martin Kvale. 2018. Bad North. (2018). Plausible Concept.
 - [32] Adam Summerville and Michael Mateas. 2016. Super Mario as a String: Platformer Level Generation Via LSTMs. *CoRR* abs/1603.00930 (2016). [arXiv:1603.00930](http://arxiv.org/abs/1603.00930) <http://arxiv.org/abs/1603.00930>
 - [33] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K. Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. 2017. Procedural Content Generation via Machine Learning (PCGML). *CoRR* abs/1702.00539 (2017). [arXiv:1702.00539](http://arxiv.org/abs/1702.00539) <http://arxiv.org/abs/1702.00539>
 - [34] Adam James Summerville, Sam Snodgrass, Michael Mateas, and Santiago Ontanon Villar. 2016. The VGLC: The Video Game Level Corpus. *Proceedings of the 7th Workshop on Procedural Content Generation* (2016).
 - [35] Adam James Summerville, Sam Snodgrass, Michael Mateas, and Santi Ontañón Villar. 2016. The VGLC: The Video Game Level Corpus. *CoRR* abs/1606.07487 (2016). [arXiv:1606.07487](http://arxiv.org/abs/1606.07487) <http://arxiv.org/abs/1606.07487>
 - [36] Vanessa Volz, Jacob Schrum, Jialin Liu, Simon M Lucas, Adam Smith, and Sebastian Risi. 2018. Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network. *arXiv preprint arXiv:1805.00728* (2018).
 - [37] Vanessa Volz, Jacob Schrum, Jialin Liu, Simon M. Lucas, Adam M. Smith, and Sebastian Risi. 2018. Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2018)*. ACM, New York, NY, USA, 8. DOI : <http://dx.doi.org/10.1145/3205455.3205517>