Proceedings of the Fifteenth AAAI Conference on Artificial
Intelligence and Interactive Digital Entertainment (AIIDE-19)

Levels from Sketches with Example-Driven Binary Space Partition

Sam Snodgrass
Northeastern University
sam.psnodgrass@gmail.com

Abstract

Procedural content generation via machine learning (PCGML) has been demonstrating its usefulness as a content and game creation approach, and has been shown to be able to support human creativity. In this paper we present an example-driven adaptation of a classic PCG approach, binary space partition (BSP), that takes a structural template or sketch of a level and fills in the details from examples. We show that this example-driven adaptation can generate a diverse set of levels from a single structural template. We evaluate the levels generated in terms of difference between paths through the levels, amount of the level copied from the examples, and other common PCG level evaluation metrics. Furthermore, we compare this method to a Markov chain approach and show that our BSP approach matches the training level distribution better while generating a greater range of interesting features.

1 Introduction

Procedural content generation (PCG) describes the algorithmic creation of content (e.g., levels, stories, quests, rules, full games, etc.). PCG via machine learning (PCGML) (Summerville et al. 2018) denotes a subgroup of PCG techniques that function by learning a model of the type of content to be generated and then sampling from that model to create new instances of the content (e.g., learn from a set of example game levels, and then generate a new level). The main challenges of PCGML approaches are that (1) they typically do not offer the user much intuitive control over the generated content (i.e., the user can only control the parameters of the model which may have unintuitive effects on the models and output); and (2) the generated content does not cover a diverse space resulting in uninteresting content.

We present a PCGML approach that offers control over the structure of the generated levels to the user while covering a diverse content space. Our approach takes a low resolution sketch of a level design (either from a user or a generated sketch) and scales up the resolution. Here we are using the term sketch to refer to a template of the structural features of the level. We compare two methods for realizing that template, a new example-driven Binary Space Partitioning (EDBSP) approach which matches structural elements between the input sketch and the example levels; and an existing mulit-layer Markov chain approach using the sketch as the input layer (adapted from our previous work on multi-layer level generation (Snodgrass and Ontañón 2017c)). Section 3 presents both of these approaches in more detail. We show that while both approaches cover a similar metric space, the distribution of levels created by our EDBSP approach more closely follows the distribution of the training levels. Furthermore, the EDBSP approach is more easily extensible to additional domains. The main contributions of this paper are:

1. A new example-driven extension to space partitioning algorithms for generating levels from a structural sketch.
2. A comparative analysis of this method and an existing PCGML approach, showing this new approach covers a larger generative space across several of the metrics

2 Related Work

There are a number of PCG approaches that generate levels or content at various resolutions. Snodgrass and Ontañón (Snodgrass and Ontañón 2017a) created a hierarchical Markov chain approach that modeled and generated levels at multiple resolutions. However, this approach required experimentally or manually tuning clustering parameters to identify the high-level structures for a given domain which can be unintuitive for non-technical users; the approach we present here only requires definitions of which game elements are solid or not. Ma et al. (Ma et al. 2014) propose an approach that takes a node graph representing a level layout, and generate a more detailed level structure using polygonal blocks provided by the researchers. Our approach differs from theirs primarily in that our approach automatically defines the level partitioning and extracts the more detailed blocks from a set of training examples. Liapis et al.'s Sentient Sketchbook (Liapis, Yannakakis, and Togelius 2014) is a mixed mixed initiative level design tool that continually refines and offers suggestions of level elements at increasing resolutions. Similarly, Guzdial et al.'s (Guzdial, Liao, and Riedl 2018) co-creative PCGML approach alternates with the user in adding elements to a shared level. These approaches assist designers in refining a given level, but we believe that our approach can be used for rapidly prototyping many variations of a base level structure.

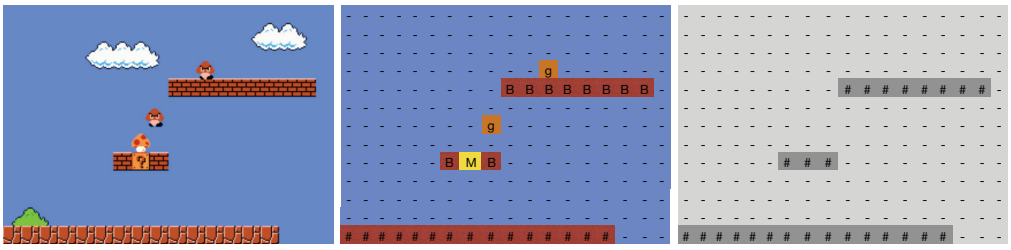


Figure 1: This figure shows a section from a *Super Mario Bros.* level (a), that same section represented with the full resolution representation (b), and that section represented with the sketch resolution representation (c).

Our approach also differs from Guzdial and Riedl's (Guzdial and Riedl 2016) level generation approach and Summerville and Mateas's (Summerville and Mateas 2016) LSTM approach in that our approach directly allows the user to guide the generator by providing a structural sketch of the level as a basis. Our previous constrained generation approach (Snodgrass and Ontañón 2016) gave control over the features of the generated levels to the user, but that control needed to be encoded into functional constraints in the source code of the model. Our approach does not offer control as directly as that approach, but the control given to the user by our approach is in a more intuitive form.

We compare our proposed approach against our previous multi-layer MdMC approach (Snodgrass and Ontañón 2017c). This approach represents levels using different tile layers corresponding to features of the levels (e.g., desired path, structures, etc.). It then learns a probabilistic relationship between nearby level elements (across layers) and the elements in the layer to be generated by the model later. This model functions in much the same way as the standard MdMC approach (Snodgrass and Ontañón 2017a), by learning probabilistic relationships between different tile types and structures, with the core difference being that the multi-layer approach contains multiple representations of a level and learns relationships across those representations. In our evaluation, we treat a level sketch (or structural template) as one of the layers and the full resolution level as the layer to be modeled for generation. Figure 1 shows a full resolution layer (b) and the sketch layer (c). More details on this model can be found in (Snodgrass and Ontañón 2017c).

3 Methods

In this section we introduce the binary space partition algorithm and our example-driven extension. Next, we describe our experimental setup, our domain, and evaluation metrics.

3.1 Example-Driven Binary Space Partition

Binary Space Partition (BSP) (Shaker et al. 2016) is a partitioning algorithm that recursively divides a given space into two smaller sections until an end condition is met. Algorithm 1 shows the procedure of the BSP algorithm. At a high level, BSP requires a space or shape to partition (line 1) and some end condition (in this case a minimum size of a section, line 2). The algorithm then splits the given section into two parts by randomly choosing a vertical or horizontal split and the position of the split (line 3), which results in two new sections (line 4). Next, if the end condition is not met (i.e., if the section could still be split, line 5) then recursively split the created sections (line 6); and similarly for the other created section, (line 8). During the splitting process, sections of various sizes will be created due to the random selection of split orientations and positions. Figure 2 (c) shows a potential splitting of a level. Once all the sections have been decided, then the algorithm needs a procedure for filling in the details of those created sections (lines 10-12).

Algorithm 1 Binary Space Partition (adapted and extended from (Shaker et al. 2016))

1: **Require:** $Level \leftarrow$ an $h \times w$ grid
2: **Require:** $min \leftarrow$ minimum size of a section
3: Divide the section along a vertical or horizontal line
4: Select one of the two newly created sections, s
5: **if** $s_{height} \geq 2 \cdot min$ or $s_{width} \geq 2 \cdot min$ **then**
6: Go to line 3 using s
7: **else**
8: Select the other section and go to line 5
9: **end if**
10: **for all** s **do**
11: Fill in the details of s
12: **end for**

The process of filling in the section details is where our work differs from traditional uses of BSP. Previous applications of BSP to level generation have focused on dungeon generation (Shaker et al. 2016; Baron 2017). These BSP level generation approaches start with empty sections and fill in the details of the sections using a manually defined set of rules or heuristics (e.g., create a room of a certain size, place items and enemies in various positions in that room, connect the rooms with hallways, etc.). In our work, we flesh out the details of the level using examples from training levels.

Figure 2 shows the flow of our approach. We start with a structural template representing the solid and empty spaces (Figure 2: b). We extract the template or sketch from a given input level (Figure 2: a), but these sketches can be created manually or generated in a number of ways. Next, we split the sketch into a set of non-overlapping sections (Figure 2: c) using the BSP (Algorithm 1: lines 1-9). To fill in the de-

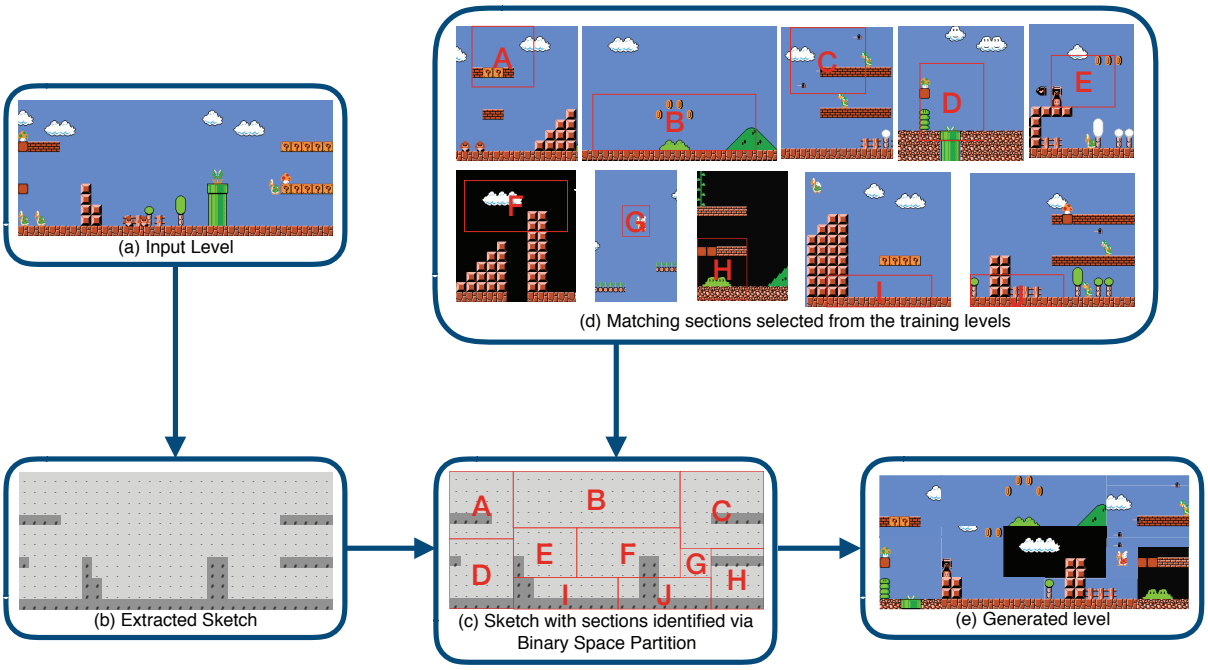


Figure 2: This figure shows the pipeline of our Example-Drive Binary Space partition (EDBSP) approach. Our approach starts with a low resolution sketch extracted from a given input level (a, b). Next, it splits the sketch into sections (c). For each of the sections our approach finds matching low resolution sections in the training levels (d). Those matching sections are then used to scale up the resolution of the sketch to a full resolution level[1](e).

tails of the sketch sections, we leverage existing levels. That is, given a set of training levels, we represent those training levels as low resolution sketches. Then for each section in the input sketch, we perform an exhaustive search over the sections of equal dimensions in the training levels and retrieve any sections that perfectly match the structure (empty and solid tile layout) of the input sketch section (Figure 2: d). We then randomly choose one of the matching sections and use the corresponding section in the full resolution version of the training level to fill in the details of the input sketch section, resulting in the final output level (Figure 2: e). In this paper we use a naive approach of performing exhaustive search for matching sections in the training levels and randomly sampling from the matching sections to fill the details. This random sampling from the matching sections encodes a distribution over possible sections because we do not remove duplicate matching sections. There are simple extensions that can be explored in this sampling process as well. For example, to avoid situations where no matching section exist, one can use an iterative loosening or set a threshold on the number of mismatches between the input sketch section and the searched sections. To increase collaboration, one can present the matching sections to the user and give them the option to choose how to fill in the details. We leave the extensions for future work and focus on the base algorithm.

[1]We show the exact image sections being stitched; but in practice we do not capture aesthetic differences, so the full resolution levels shown later do not contain disparity in visuals.

3.2 Domain

We test our approach in *Super Mario Bros.*, a platforming game that has been extensively explored by the PCG (Shaker et al. 2011; 2012; Togelius et al. 2009) and PCGML research communities (Snodgrass 2018; Summerville and Mateas 2016; Guzdial and Riedl 2016; Dahlskog, Togelius, and Nelson 2014; Volz et al. 2018) . We use a set of 30 levels from *Super Mario Bros.* and *Super Mario Bros.: The Lost Levels* that includes the standard outside levels (i.e., not castle, underground, underwater, or mushroom top levels). We represent levels using grids where each tile corresponds to a position in the level, and each tile take a value from a representative tile set (i.e., tile types) as is common in the literature (Snodgrass 2018; Summerville and Mateas 2016). We use two sets of tile types for different resolutions:

- **Full Resolution:** This uses 35 tile types representing the various level structures (e.g., solid bricks, pipes, moving platforms), interactive elements (e.g., breakable blocks, ?-blocks, springs), different enemy types (e.g., koopas, hammer bros., goombas), and empty space. Figure 1 (center) shows a section of a level represented in this format.

- **Sketch Resolution:** This uses 2 tile types that distinguish between the solid elements (e.g., pipes, blocks, springs, platforms) and non-solid elements of the level (e.g., empty space, enemies, flag poles). Figure 1 (right) shows a section of a level represented in this format.

3.3 Experimental Evaluation

We evaluate our example-driven binary space partition (EDBSP) approach in a comparative analysis with the multi-layer MdMC approach (Snodgrass and Ontañón 2017c). We follow some of Summerville's PCG evaluation recommendations (Summerville 2018), including performing KDE visualizations, and presenting representative generated levels.

For our experiments we employ the following procedure. First, we represent the set of 30 levels using both the full and sketch resolutions. For each level, we use the sketch as input to our EDBSP approach and to the multi-layer MdMC approach. We use the other 29 levels as the training data for the MdMC approach and as the source of examples for the EDBSP approach. We do not use the current input sketch level as training data in order to show that the method generalizes to sketches outside of the training set. We then generate 100 full resolution levels for the current sketch with our EDBSP approach and the MdMC approach.

To evaluate our approach and the multi-layer MdMC approach, we compute the metrics described below for all the generated levels. We then perform a kernel density estimation (KDE) (Rosenblatt 1956) on pairwise combinations of those metrics. We evaluate the similarity of the distributions of the generated level sets in the metric space to the distribution of the training levels in the metric space using the e-distance (Székely and Rizzo 2013), as suggested by Summerville (Summerville 2018). Lastly, we present the generated levels which copied the most from the training levels.

The metrics used in the KDE and e-distance measure are described below. We are interested in modeling and generating variations on the training level structures, and therefore desirable values for our metrics used to compute the e-distance are those values that fall around the values of those metrics on the training level set.

- **Leniency (Smith and Whitehead 2010):** This is meant to approximate the difficulty of a level in terms of the frequency of detrimental elements (i.e., enemies and gaps) and beneficial elements (i.e., power-ups) present in the level normalized over the length of the level. This is computed as: $\frac{.5 \times powerups - (gaps + .5 \times enemies)}{length}$
- **Interesting Tile Frequency:** This captures how frequently tiles other than empty or solid appear in the level.
- **Path Coverage Frequency:** This measures the percentage of the level that is covered by an A^* agent's path.
- **Meaningful Jumps Count:** This counts how many times an A^* agent *needed* to jump to complete the level. The agent may jump many times, but this only counts jumps that allow the agent to cross a gap or avoid an enemy.
- **Fréchet Distance (Snodgrass and Ontanón 2017b; Eiter and Mannila 1994):** This measures the distance between two paths. Intuitively it can be thought of as the minimum length of a rope needed to connect two people walking on two separate paths. Specifically, in our work we compute the distance between the path through a generated level and the path through the original level used as a template. This metric can help indicate if the differences between the generated and template levels are impactful to the gameplay. Note that this metric is not used in the e-distance calculation because this is a pairwise metric that is not computed on only the training levels.
- **Plagiarism:** This metric compares the generated levels against the training levels and finds the largest section of columns in the generated level that has an exact match in the training levels. This helps us investigate the novelty of the generated levels. Note that this metric is not used in the e-distance calculation because this is a pairwise metric that is not computed on only the training levels.

Sketch	EDBSP and Training	MdMC and Training	EDBSP and MdMC
1	34.350	63.965	28.351
2	34.267	76.335	108.897
3	50.217	46.600	4.068*
4	21.454	49.258	*157.613*
5	49.367	**23.114**	60.490
6	**18.740**	51.057	132.359
7	47.413	41.756	4.777*
8	40.065	37.823	9.415*
9	53.173	79.533	41.324
10	137.722	157.038	3.412*
11	*548.693*	*543.087*	23.282
12	120.978	131.591	2.001*
13	57.199	27.183	40.781
14	26.596	43.061	26.398
15	54.383	52.782	9.765
16	76.652	59.424	41.707
17	34.385	42.789	23.772
18	29.390	39.131	54.229
19	40.089	100.847	43.996
20	106.365	78.932	12.912*
21	58.823	33.930	55.727
22	32.384	32.181	**1.929***
23	192.885	241.848	15.528
24	20.237	59.934	94.719
25	33.087	35.640	3.457*
26	82.501	72.183	7.774*
27	26.070	43.140	12.446
28	28.273	33.321	3.597*
29	41.224	45.764	**1.929***
30	98.115	170.891	32.700
Avg	73.170	83.805	35.312

Table 1: This table shows the e-distances between the generated levels for each input sketch and the training levels (middle two columns) and the e-distance between the levels generated for each input sketch by each model (last column). **Bold** indicates the generated levels with the lowest e-distances; *italics* indicate the highest e-distance for each model, and an * indicates that the distributions do not have a statistically significant difference (i.e., all unmarked distances between distributions are statistically significant using $p = 0.05$ with a homogeneity test on the distributions).

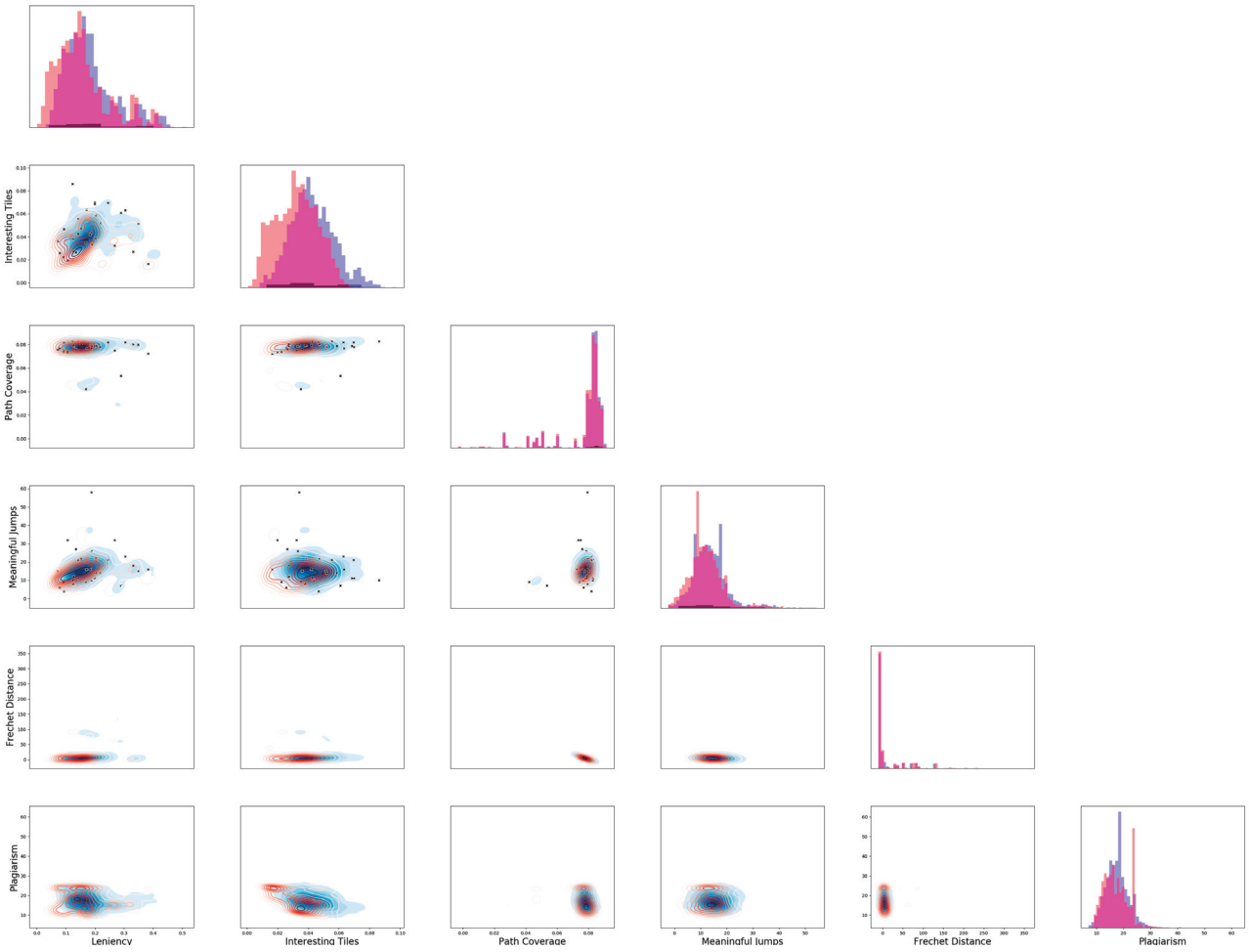


Figure 3: This figure shows a corner plot of the kernel density estimation plots for our chosen metrics on the levels generated by the Example-driven BSP approach (Blue), the multi-layer MdMC approach (Red), and the training levels in cases where those metrics are defined on the training levels (Black).

4 Results

Figure 3 shows a corner plot of the kernel density estimation (KDE) plots for the metrics described in the previous section for all the levels generated with our EDBSP approach (blue shades), generated with the multi-layer MdMC approach (red outlines), and the training levels for the applicable metrics (black points). For most metrics and plots the MdMC and EDBSP approaches cover similar spaces. However, for the *Interesting Tile Frequency* we can see that our EDBSP approach is skewed towards higher values than the MdMC approach. This is likely due to the probabilistic nature of the MdMC approach which learns relationships between tile types, and is therefore more likely to sample common tiles (e.g., empty space and solid blocks in *Super Mario Bros.*), whereas the EDBSP approach randomly grabs a matching section without accounting for tile-level relationships. This is interesting because it shows that even if both approaches cover otherwise similar metric spaces, the EDBSP approach will be more visually and interactively diverse.

Table 1 shows the e-distance between the distribution of the generated levels for each input sketch and the training levels, and between the distributions of the generated levels for each input sketch with each model using the *Interesting Tile Frequency*, *Path Coverage*, and *Meaningful Jumps* metric space. The e-distances between the distributions are statistically significant (with $p = 0.05$ using a homogeneity test) unless noted with an asterisk (*). We see that the EDBSP generated level sets generally have a smaller distance from the training level distribution, indicating that the EDBSP approach more accurately captures the features of the training data than the MdMC approach. There is admittedly not a large difference between the e-distance values of the models from the training sets. This is also seen in the e-distances between the sets of generated levels, so we will now highlight the interesting results in the table.

It is important to notice that the levels generated for sketch 11 by both models have a much larger e-distance from the training level distribution than any of the other generated

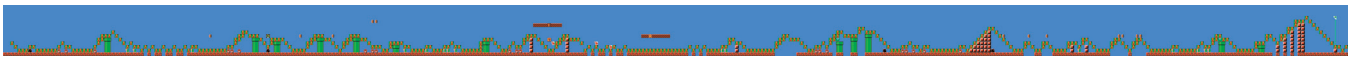
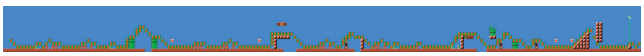
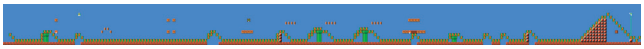


Figure 4: This figure shows the 11^{th} training level and a path through that level.



(a) The 6^{th} training level and a path through that level.

(b) Level generated by the EDBSP approach with the 6^{th} sketch. *Plagiarism* of 27 columns.



(c) The 5^{th} training level and a path through that level.

(d) Level generated by the MdMC approach with the 5^{th} sketch. *Plagiarism* of 24 columns.

Figure 5: This figure shows the sketches for the level sets that had the lowest e-distance from the training levels (a, c) and the levels generated by the respective approaches with those sketches with the highest plagiarism values (b, d).

sets. By looking at sketch 11 (Figure 4) we can see that there are many gaps which both requires many meaningful jumps and increases the leniency of the level. This, in turn, results in generated levels having a higher number of meaningful jumps and higher leniency than the rest of the training levels.

Next, we look at the generated level sets with the lowest e-distance scores from the training set. Figure 5 shows the levels used as the sketches and the generated levels for those sets with the largest number of plagiarized columns. For the EDBSP approach, though there are 27 plagiarized columns (near the end of the level), there are still interesting differences from the input level. Specifically, the overhang area towards the middle of the generated level has a new interactive area, and the end of the level includes an additional obstacle (the downward pipe). Alternatively, the generated MdMC level copies 24 columns (also near the end), but instead of introducing interesting new interactions it removes interactivity by replacing ?-blocks with solid ones and pipes with pillars. This reinforces the point regarding the difference in frequency of interesting tiles between the methods.

In addition to the distance of the generated levels from the training levels, we can also see the distance between the sets of generated levels themselves. Here we can see that many of the generated sets are more similar to each other than to the training set (which is reflected in KDE projections as well in Figure 3). A notable exception occurs in the levels generated using sketch 4. Here, the e-distances between the generated level sets are much larger than the e-distances between the individual generated sets and the training set. This is likely due to the large flat areas in the training level. This results in the MdMC model filling these areas with solid and empty tiles (with few enemies or obstacles) and the EDBSP model filling in those spaces with more enemies. This causes the set of generated EDBSP levels to have more meaningful jumps, a higher leniency, and higher path coverage than the MdMC generated levels, which in turn results in a larger e-distance.

5 Discussion, Conclusions, and Future Work

We have shown that our example-driven Binary Space Partition (EDBSP) and the multi-layer (MdMC) were capable of covering similar spaces in the chosen metric space with a key difference in the frequency of interesting tiles. This difference displayed itself as we explored generated levels with more plagiarism and saw that the EDBSP-generated levels inserted more interesting and interactive areas whereas the MdMC-generated levels removed such areas and replaced them with solid structures or sparse areas. In future work it would be interesting to explore player perceptions of the interestingness of the levels and the amount of interactivity between the approaches. Additionally, the metrics we employed for the KDE and e-distance were skewed towards pathing information, but it would be worthwhile to explore more metrics for representing the aesthetics and interestingness of the levels for these evaluations, particularly because the structural elements are common across the levels. The key takeaway here is that while both models performed similarly for many of the metrics, the EDBSP model retained more of the interesting level structures and patterns.

There are several avenues to explore moving forward. We have shown this method to work in *Super Mario Bros.*, but by encoding levels from any number of domains in a solid or empty sketch format we can blend, merge, and extend the approach to many new domains. Including levels from more domains into the examples may have interesting effects on the output, and could increase diversity in the generated levels. Our goal for this method is to enable rapid prototyping of game levels; therefore, an obvious avenue is a user evaluation of a mixed-initiative tool for level design that employs this method. A human evaluation of the usability, clarity, and controllability of the model is an important next step. A part of this user study is allowing designers to provide new level sketches for the algorithm to flesh out. Lastly, we've shown our approach in 2D levels, but this approach should extend to any space partitioning approach in 2D or 3D. Developing and evaluating these extensions in more complex domains and with more complex partitioning approaches can showcase the usefulness and generality of the underlying ideas.

In this paper we presented an example-driven extension to a classic PCG approach, binary space partition. We evaluated our extension by performing a comparative analysis with another PCGML approach. We have shown that our EDBSP approach generates levels with more interesting elements while achieving similar performance across other metrics. Moving forward we want to explore the usability of the model in a collaborative setting.

References

Baron, J. R. 2017. Procedural dungeon generation analysis and adaptation. In *Proceedings of the SouthEast Conference*, 168–171. ACM.

Dahlskog, S.; Togelius, J.; and Nelson, M. J. 2014. Linear levels through n-grams. *Proceedings of the 18th International Academic MindTrek*.

Eiter, T., and Mannila, H. 1994. Computing discrete Fréchet distance. Technical Report 94/64, Technische Universität Wien.

Guzdial, M., and Riedl, M. 2016. Game level generation from gameplay videos. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Guzdial, M.; Liao, N.; and Riedl, M. 2018. Co-creative level design via machine learning. In *2018 Experimental AI in Games Workshop*.

Liapis, A.; Yannakakis, G. N.; and Togelius, J. 2014. Designer modeling for sentient sketchbook. In *2014 IEEE Conference on Computational Intelligence and Games*, 1–8. IEEE.

Ma, C.; Vining, N.; Lefebvre, S.; and Sheffer, A. 2014. Game level layout from design specification. In *Computer Graphics Forum*, volume 33, 95–104. Wiley Online Library.

Rosenblatt, M. 1956. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics* 832–837.

Shaker, N.; Togelius, J.; Yannakakis, G. N.; Weber, B.; Shimizu, T.; Hashiyama, T.; Sorenson, N.; Pasquier, P.; Mawhorter, P.; Takahashi, G.; et al. 2011. The 2010 mario AI championship: Level generation track. *TCIAIG, IEEE Transactions on* 3(4):332–347.

Shaker, N.; Nicolau, M.; Yannakakis, G. N.; Togelius, J.; and O'neill, M. 2012. Evolving levels for super mario bros using grammatical evolution. In *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, 304–311. IEEE.

Shaker, N.; Liapis, A.; Togelius, J.; Lopes, R.; and Bidarra, R. 2016. Constructive generation methods for dungeons and levels. In Shaker, N.; Togelius, J.; and Nelson, M. J., eds., *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer. 31–55.

Smith, G., and Whitehead, J. 2010. Analyzing the expressive range of a level generator. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, 4. ACM.

Snodgrass, S., and Ontañón, S. 2016. Controllable procedural content generation via constrained multi-dimensional Markov chain sampling. In *25th International Joint Conference on Artificial Intelligence*.

Snodgrass, S., and Ontañón, S. 2017a. Learning to generate video game maps using Markov models. *IEEE Transactions on Computational Intelligence and AI in Games*.

Snodgrass, S., and Ontanón, S. 2017b. Player movement models for platformer game level generation. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, 757–763. AAAI Press.

Snodgrass, S., and Ontañón, S. 2017c. Procedural level generation using multi-layer level representations with mdmcs. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, 280–287. IEEE.

Snodgrass, S. 2018. *Markov Models for Procedural Content Generation*. Drexel University.

Summerville, A., and Mateas, M. 2016. Super Mario as a string: Platformer level generation via LSTMs. *Proceedings of 1st International Joint Conference of DiGRA and FDG*.

Summerville, A.; Snodgrass, S.; Guzdial, M.; Holmgård, C.; Hoover, A. K.; Isaksen, A.; Nealen, A.; and Togelius, J. 2018. Procedural content generation via machine learning (PCGML). *IEEE Transactions on Games*.

Summerville, A. 2018. Expanding expressive range: Evaluation methodologies for procedural content generation. In *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Székely, G. J., and Rizzo, M. L. 2013. Energy statistics: A class of statistics based on distances. *Journal of statistical planning and inference* 143(8):1249–1272.

Togelius, J.; Karakovskiy, S.; Koutník, J.; and Schmidhuber, J. 2009. Super mario evolution. In *2009 ieee symposium on computational intelligence and games*, 156–161. IEEE.

Volz, V.; Schrum, J.; Liu, J.; Lucas, S. M.; Smith, A.; and Risi, S. 2018. Evolving mario levels in the latent space of a deep convolutional generative adversarial network. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 221–228. ACM.