Graph Grammars for Super Mario Bros Levels

Santiago Londoño LONDONO@CS.UNI-BONN.DE
Olana Missura OLANA.MISSURA@UNI-BONN.DE
University of Bonn, Germany

Abstract

We aim to automatically create for a platformer game levels structurally similar to provided training levels. We assume that (i) the structure of a level in a platformer game has a profound influence on the enjoyment of players, and (ii), given levels created by expert designers, it is possible to extract their structural properties and transfer them to new levels. In this paper we first propose a graph-based representation of *Super Mario Bros* levels to encode their structure. Next, to abstract the structural elements, we extend an algorithm for learning a graph grammar, SUBDUEGL, to produce a stochastic graph grammar. We describe our work in progress on generating new levels from graphs produced using the inferred graph grammar.

1. Introduction

Conventionally, specialized designers build game levels, applying their creativity and expert knowledge to produce content exhibiting structural correctness, interestingness for players, and balanced difficulty among other quality features. We aim to create a learning mechanism capable of extracting design concepts from a set of human-authored levels and to use the acquired knowledge to algorithmically generate new ones. To that purpose we propose to use stochastic graph grammars in their both qualities, descriptive and generative.

2. Related work

Diverse techniques have been used to automatically generate content for computer games since the early eighties (Hendrikx et al., 2013). Most of these applications have the same general objective: to *autonomously* create game elements. Level generation in particular, has been explored through a variety of approaches.

Search-based techniques, such as evolutionary algorithms, genetic programming and other forms of stochastic optimization have been employed in multiple ways, e. g. by Cardamone et al. (2011) and Johnson et al. (2010).

Specifically for *Super Mario Bros*, Snodgrass & Ontañón (2014) abstracted levels as matrices and randomly generated new ones through Markov Chains. This representation is too simplistic, though. It is the interactions of various elements of a level that defines its quality and these interactions are too complex to be analyzed directly over a matrix.

There are a few works using grammars to represent levels. Dormans (2010) builds level layouts for adventure games over mission scripts, based on context-free string and space grammars. Shaker et al. (2012) generate *Super Mario Bros* levels using genetic programming to search the space of levels derivable from a predefined grammar, an approach that has been called Grammatical Evolution. In this work we go a couple of steps further by using graph grammars to encode the relationships between various elements of a level and by learning these grammars instead of requiring them to be predefined.

To the best of our knowledge, graph grammars have not been applied as yet to algorithmically generate game contents. Nevertheless, they have shown promising results in diverse fields, for instance for discovering interesting patterns in chemical compounds (Cook & Holder, 1994) or in the execution of programs (Zhao et al., 2008).

3. Representation of levels

From an implementation perspective, *Super Mario Bros* levels are represented as a matrix M of bytes, with dimensions $width \times height$, where each entry corresponds to a *sprite*. In contrast, from an analytical point of view, *Super Mario Bros* levels are composed of elements that are not isolated, but relate to each other in different ways. The matrix M contains all this information, but in a way that is convoluted and hard to analyze. An explicit and succinct

Preliminary work. Under review by the Constructive Machine Learning workshop @ ICML 2015. A version of this paper has been submitted to the workshop on Procedural Content Generation workshop @ FDG 2015. Do not distribute.

representation of the levels is fundamental for the development of effective algorithms for learning and procedurally generating correct content. Hence, we suggest to use graphs as a representation of levels.

3.1. Levels as graphs

We represent *Super Mario Bros* levels as *directed graphs* $G = \langle V, E \rangle$, with V a set of *nodes* and $E \subseteq V \times V$ a set of *edges*. Both nodes and edges have labels in the sets Δ and Δ_E respectively. The set Δ is made up of the names of all the game elements of interest, (e. g. blocks, coins, etc.) The set Δ_E comprises the types of relationships between these elements.

One of the most significant advantages of using graphs, is that edges enable us to establish semantic relationships between the elements of the levels. We use edges to define two classes of elements essential to our graph representation:

Platforms are groups of solid sprites that are consecutive and lie on the same row of the level matrix. They form solid positions upon which the character can stand and jump to other elements.

In a level graph, platforms are represented by edges, labeled as *platform* that connect their first and last sprites, represented as solid nodes. In addition, platform edges have a *length* attribute, stating the number of sprites it spans. Here, we denote platforms by $p = (p_s, p_t) \in V \times V$, where p_s and p_t are the start and end nodes.

Item clusters are subgraphs comprising a set of neighboring nodes, denoted by $C = \{c_1, ..., c_n\} \subseteq V$. They represent conglomerates of sprites that are not solid, but still interact with the player (e. g. coins). Each node in an item cluster is connected with its closest neighbor (i. e. another sprite located at a distance lower than a threshold), through an edge labeled as *cluster*.

Moreover, the concept of *reachability*, fundamental to our approach, is described in terms of a relationship between nodes.

Let $G = \langle V, E \rangle$ be a graph representing a level grid M and $p_x \in V$ a node that is part of a platform. We define the ***reachability pyramid*** of p_x, as the isosceles triangle having p_x at the center of its base and spanning the area of M directly above it.

Now, let $v \in V$ be a node and $p = (p_s, p_t) \in V \times V$ a platform. The node v is said to be ***reachable*** from p, if and only if the position of v in M is contained in the reachability pyramid of p_s or p_t, or the rectangle delimited by the top of these pyramids and by p_s and p_t. Figure 1 illustrates these ideas.

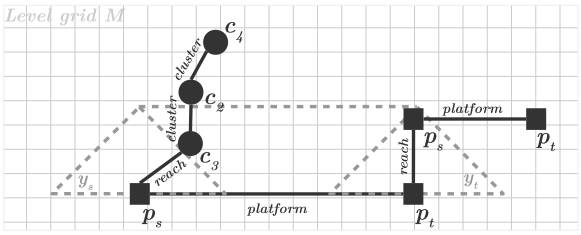

Figure 1. Reachability area of a platform, delimited by the dashed lines. The triangles y_s and y_t depict the reachability pyramids of the start and end nodes.

Finally, let $p = (p_s, p_t),\ q = (q_s, q_t) \in V \times V$ be two platforms. We say that q is reachable from p, if and only if either (or both), q_s and q_t are reachable from p.

In the graph abstraction, reachability relationships are represented as edges with label *reach*.

3.2. Transformation of level grids into graphs

In order to obtain the graph representation of the example levels we will learn from, we implemented an algorithm to transform a level in matrix form M, into a graph G, composed of interconnected platforms and item clusters.

The algorithm consists of two major stages:

Detection of elements: platforms and item clusters are detected by scanning the level grid M row by row. *Platforms* are built by grouping adjacent sprites, found at the same row and that are suitable to be part of a platform.

Item clusters are constructed through a simplified version of the GDBSCAN algorithm (Sander et al., 1998). They are built up starting at a non-solid, interactive sprite and expanding the cluster over its neighborhood.

Assembly of reachability edges: once all platforms and item clusters in the map have been constructed, the algorithm evaluates which elements are reachable from each platform and inserts *reach* labeled edges accordingly.

Level grids are built from graphs in two stages: First, the main structure of the level is constructed, by rendering the elements of the graph on the grid. Second, adornments such as background sprites are randomly added around the main structure.

4. Learning from level graphs

We hypothesize that human-authored, high-quality levels contain design paradigms reflecting the designer's knowledge, encoded in a way representable by graphs.

Under these assumptions, graph data mining techniques can be applied to extract the knowledge embedded in a set

of human-authored levels. Specifically, we implemented the SUBDUEGL algorithm (Jonyer et al., 2004), which learns a graph grammar from a set of example graphs, striving to achieve an optimal balance between the size of the substructures and their frequency of occurrence. This criterion is called the *MDL principle* (Cook & Holder, 1994). The induced graph grammars are expected to define patterns frequently observed in high-quality levels.

4.1. Node Label Controlled graph grammars

SUBDUEGL learns Node Label Controlled (NLC) graph grammars (Rozenberg & Ehrig, 1999), which are context-free by definition and have the form:

$$R = (\Sigma, \Delta, P, C, S) \ ,$$

where Σ is the alphabet of terminal and non-terminal node labels, Δ is the alphabet of terminals, P is the set of productions, C is the set of connection relations, S is the initial graph. A single production from P has the form $z \rightarrow \alpha$, with $z \in \Sigma \setminus \Delta$ a non-terminal node label and α a graph, i. e. it defines how a node with a non-terminal label can be replaced by a subgraph α. A connection relation is an ordered pair $(s, t) \in \Delta \times \Delta$.

The *node label controlled* aspect of R, implies that α would be connected to the neighborhood of the replaced node by means of a *connecting mechanism* (Rozenberg & Ehrig, 1999). For a production rule $z \rightarrow \alpha$ it operates by adding an edge from a node labeled s in the neighborhood of z, to a node labeled t in α, for each *connection instruction* $(s, t) \in C$.

4.2. Stochastic NLC graph grammars

To accurately represent the occurences of found substructures, we extend SUBDUEGL to learn stochastic NLC graph grammars. An example is shown in Figure 2.

A *stochastic NLC graph grammar* G, has "probabilities of being applied" $p_{z,1}, ..., p_{z,n_z}$ associated to its productions $z \rightarrow \alpha_1, ..., z \rightarrow \alpha_{n_z}$, in such a way that for a particular left-hand side z, $\sum_{j=1}^{n_z} p_{z,j} = 1$.

As proposed by Oates et al. (2003), a maximum-likelihood estimate for the probability of application of a production $z \rightarrow \alpha$, provided a set of example graphs $\mathbf{G_X} = \{G_1, ..., G_m\}$, can be computed as:

$$\hat{p}\left(z \rightarrow \alpha\right) = \frac{c\left(z \rightarrow \alpha | \mathbf{G_X}\right)}{\sum_{\delta} c\left(z \rightarrow \delta | \mathbf{G_X}\right)}$$

For each substructure α_i discovered by SUBDUEGL in the set of examples $\mathbf{G_X}$, a production $P = z_i \rightarrow \alpha_i$ is synthesized. Thus, the total number of times each production was observed, namely $c(z \rightarrow \alpha | \mathbf{G_X})$, can be computed as a byproduct. Suppose that SUBDUEGL is run on a source graph $G_m \in \mathbf{G_X}$ and as a result, a set of substructures $\mathbf{B} = \{\alpha_i : i \in [1, k] \wedge \alpha_i \subseteq G_m\}$ are chosen to become productions. Each substructure appears $c\left(\alpha_i\right)$ times in the input graph. The resulting productions are:

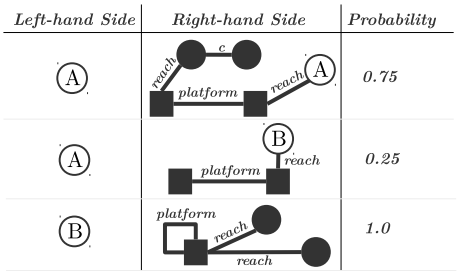

Figure 2. A stochastic graph grammar.

$P = \{z \rightarrow \alpha_i : i \in [1, k] \wedge \alpha_i \subseteq G_m\}$, with z non-terminal

Their associated probabilities are computed as:

$$\hat{p}\left(z \rightarrow \alpha_i\right) = \frac{c\left(\alpha_i\right)}{\sum_{\delta \in \mathbf{B}} c\left(\delta\right)}$$

5. Level generation

The level generation algorithm will first create a new graph using the learned grammar: It will expand an initial graph (e.g. a non-terminal node representing the start of the level) by iteratively applying the productions of the stochastic NLC graph grammar. On each iteration, the algorithm expands a non-terminal node with label z. To do so, it selects a production $z \rightarrow \alpha$, in accordance with the probabilities having z at the left-hand side. In the next step, the so generated graph is transformed into a level grid and additional details such as background sprites will be rendered on it.

6. Current status of the project

We have implemented the transformation of level grids into graphs and a stochastic version of SUBDUEGL, as specified in Section 4. The level generation process is yet to be completed.

To evaluate the quality of our results, we intend to use the metrics proposed by Shaker et al. (2012). Furthermore, we would design an experiment involving human players to qualitatively evaluate the generated levels and to compare our system against existing level generators, namely (i) the *default generator* included as a part of the framework, (ii) the *GE generator* by Shaker et al. (2012), based on grammar evolution; (iii) the *winner of the Mario AI Championship 2012*, an approach driven by the player's score, as presented in Togelius et al. (2013).

References

Cardamone, Luigi, Yannakakis, Georgios N., Togelius, Julian, and Lanzi, Pier Luca. Evolving interesting maps for a first person shooter. In *Applications of Evolutionary Computation*, pp. 63–72. Springer, 2011.

Cook, Diane J. and Holder, Lawrence B. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, pp. 231–255, 1994.

Dormans, Joris. Adventures in level design: generating missions and spaces for action adventure games. In *Proceedings of the 2010 workshop on procedural content generation in games*, pp. 1. ACM, 2010.

Hendrikx, Mark, Meijer, Sebastiaan, Van Der Velden, Joeri, and Iosup, Alexandru. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, 9(1):1, 2013.

Johnson, Lawrence, Yannakakis, Georgios N., and Togelius, Julian. Cellular automata for real-time generation of infinite cave levels. *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, pp. 1–4, 2010. doi: 10.1145/1814256.1814266.

Jonyer, Istvan, Holder, Lawrence B., and Cook, Diane J. MDL-based context-free graph grammar induction and applications. *International Journal on Artificial Intelligence Tools*, 13(01):65–79, 2004.

Oates, Tim, Doshi, Shailesh, and Huang, Fang. Estimating maximum likelihood parameters for stochastic context-free graph grammars. In *Inductive Logic Programming*, pp. 281–298. Springer, 2003.

Rozenberg, Grzegorz and Ehrig, Hartmut. *Handbook of graph grammars and computing by graph transformation*, volume 1. World scientific Singapore, 1999.

Sander, Jörg, Ester, Martin, Kriegel, Hans-Peter, and Xu, Xiaowei. Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications. *Data mining and knowledge discovery*, 2(2):169–194, 1998.

Shaker, Noor, Nicolau, Miguel, Yannakakis, Georgios N., Togelius, Julian, and O'Neill, Michael. Evolving levels for Super Mario Bros using grammatical evolution. In *Proceedings of the IEEE Conference on Computational Intelligence and Games*, pp. 304–311. IEEE, 2012.

Snodgrass, Sam and Ontañón, Santiago. Experiments in map generation using Markov chains. In *Proceedings of the 9th International Conference on Foundations of Digital Games*, volume 14, 2014.

Togelius, Julian, Shaker, Noor, Karakovskiy, Sergey, and Yannakakis, Georgios N. The Mario AI championship 2009-2012. *AI Magazine*, 34(3):89–92, 2013.

Zhao, Chunying, Ates, Keven, Kong, Jun, and Zhang, Kang. Discovering program's behavioral patterns by inferring graph-grammars from execution traces. In *Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence*, volume 2, pp. 395–402. IEEE, 2008.