Search space optimisation for procedural content generators

Harald De Bondt

Supervisors: Prof. dr. ir. Sofie Van Hoecke, Prof. dr. Peter Lambert
Counsellors: Jelle Van Campen, Gaétan Deglorie

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Department of Electronics and Information Systems
Chair: Prof. dr. ir. Rik Van de Walle
Faculty of Engineering and Architecture
Academic year 2015-2016

Search space optimisation for procedural content generators

Harald De Bondt

Supervisors: Prof. dr. ir. Sofie Van Hoecke, Prof. dr. Peter Lambert
Counsellors: Jelle Van Campen, Gaétan Deglorie

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Department of Electronics and Information Systems
Chair: Prof. dr. ir. Rik Van de Walle
Faculty of Engineering and Architecture
Academic year 2015-2016

Foreword

I would like to express my sincere gratitude to my advisor Gaétan Deglorie for the useful comments and academic learning process, but especially for the encouraging meetings throughout the last months of the master thesis.

Besides my advisor, I would like to thank the rest of my thesis committee: professor Sofie Van Hoecke and professor Peter Lambert, whom allowed me to pursue the topic I proposed.

I would like to thank family and friends, to offer me emotional and appetizing support.

Last but not least, I would like to thank Rian and Mary-Jane for providing fresh perspective on the subject.

Harald De Bondt, May 2016

Permission of use on loan

Harald De Bondt, May 2015

Search space optimisation for procedural content generators

by

Harald De Bondt

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Academic year 2015-2016

Supervisors: Prof. dr. ir. textscSofie Van Hoecke, Prof. dr. textscPeter Lambert
Counsellors: Jelle Van Campen, Gaeétan Deglorie

Faculty of Engineering and Architecture
Department of Electronics and Information Systems
Chair: Prof. dr. ir. Rik Van de Walle

Abstract

The gaming market grows every year. In order to meet this demand, the production of games needs to scale, including the creation of its content, without hurting the games quality. The problem is that the current techniques for creating content still require a large amount of manual labour. The research domain of procedural content generation (PCG) offers a solution by automatically generating content. However, solutions from the domain often suffer from two big issue. (1) Expressiveness, the ability to generate a large amount of diverse content which is not regarded as repetitive by the consumer. (2) Controllability, the ability of the user to understand the behaviour of input parameters of the solution and their influence on the variety of generated content. This paper proposes a new solution, called the "hyper-generator" , which is made out of two process , (1) the generative process and (2) the learning process. The generative process offers high expressiveness and controllability by combining two distinct types of control, bottom-up and top-down, in a flexible way. The main components of the generative process are a user-defined probabilistic model (bottom-up) and a set of constraints on the content(top-down). The constraints influence the probabilistic model through the learning process, which adapts the model automatically for faster generation of content fitting these constraints. This contrasts with other PCG approaches, whom exclude by design possibly interesting varieties in the content in favour of reduced computation time. The expressive range of the approach is extensively evaluated using a new measurement standard based on the visualisation of probabilistic distributions. Five generalised scene layouts from architectural context have been extensively evaluated. Three of them have been successfully optimised.

Keywords

PCG,learning,controllability,expressiveness

Search space optimisation for procedural content generators

Harald De Bondt

Supervisor(s): Prof. Dr. Ir. Sofie Van Hoecke, Prof. Dr. Peter Lambert, Counsellors: Jelle Van Campen and Gaeétan Deglorie

Abstract— **The gaming market grows every year and in order to meet this demand, game content production needs to be able to scale. The domain of procedural content generation (PCG) offers a solution by automatically generating content. Current PCG approaches often suffer from two big issue. (1) Expressiveness, the ability to generate a large amount of diverse content which is not regarded as repetitive by the consumer. (2) Controllability, the ability of the user to understand the behaviour of input parameters and their influence on the variety of generated content. This paper proposes a new approach, called a "hyper-generator", made out of two process' , (1) the generative process and (2) the learning process. The generative process offers high expressiveness and controllability by combining two distinct types of control, bottom-up and top-down, in a flexible way. Other PCG approaches, exclude by design, possibly interesting varieties in the content in favour of reduced computation time. In contrast, the learning process of the hyper-generator instead tries to increase performance by optimising the probabilistic model for suitable content. The expressive range of the approach is extensively evaluated using a new metric based on visualisation of probabilistic densities. Five generalised scene layouts from architectural context have been extensively evaluated. Three of them have been successfully optimised.**

Keywords—**PCG, learning, controllability, expressiveness**

I. Introduction

The global gaming market is one of the fastest growing markets in the world with a revenue of 83.6B $ in 2014 and a forecasted 91.5B $ for 2015 [1]. The games industry is gaining in popularity and scale and with this comes a demand for games with diverse content. As the content production is becoming too expensive [2] and unscalable [3], we are facing a "content creation problem" . Automatic means of generating content - as found in the academic domain of procedural content generation (PCG) - could help resolve the content creation problem [4]. Not only can generators alleviate parts of manual authoring, they can also increase a game's replayability and unpredictability by presenting the user with a unique variety of the content. In literature, an often used metric for the contents variety is expressive range [5].

To meet the demand of more unique and complex graphics, the techniques driving generators have increased in complexity over the years. This makes them hard to use for the people in charge of content creation, for example non-experts that are not specifically trained to understand the behaviour of the underlying techniques. In current literature on PCG there are two distinct approaches for controlling over content generators, a bottom-up and top-down approach. A bottom-up approach requires the user to manipulate either low-level input parameters or the sequence of operations in the generation process. A top-down approach provides the user with a higher level of control by generating content iteratively and search for the most fitting result across generations. In order to increase controllability, both types of control should be combined in such a way that its respective disadvantages are minimized, without hurting the control type's useful characteristics or reducing the content generators expressive range. I will apply a learning algorithm to automatically optimise the search space of the iterative search process, without hurting the diversity in the search space. This reduces the number of required iterations to find the desired content and allows the use to declare the search space in less detail.

H. De Bondt is with the Computer Science Engineering, Ghent University (UGent), Gent, Belgium. E-mail: harald.debondt@UGent.be .

II. Related work

In PCG literature controllability has been referred to as how much a user understands the shift in resulting content relative to changes of the input parameters [6]. According to [5] there is a trade-off between controllability and expressiveness. A larger expressive range can require the user to tweak more content parameters, a larger range of possible values for these parameters or both. I will distinguish related work based on the techniques used to solve the problem of controllability. The chapter contains a discussion of the previous work in two domains: controllability of PCG and learning techniques applied to design problems in general.

A. Controllability in Procedural content generators

Constructive approaches, for example generative grammars, are a powerful way to describe an infinite amount of possible designs in a bottom-up way. A grammar for generating buildings is CGA++ [7] which in contrast to other grammar based techniques allows context sensitive tasks. However, Its formal, often textual, representation and recursive control logic is generally inadequate for artists. Other solutions try to avoid the text-based construction and increase controllability by using visualisation, more user-interaction or both. Visualising the internal structure of a generator can help the user better understand the generators configuration[8]. Tracery [9], enables novices to work with grammars that generate textual stories. They combine visualisation with iterative feedback by regenerating the story on edits. visualisation and user-interaction can speed up the design process and make it more intuitive, however the user still needs to manipulate most details of the content by hand.

Declarative modelling helps the designer to enforce consistency over the content properties in a top-down manner without having to worry about the details of each separate property. In [10]

they research the declarative modelling for scenes. They divide the scene modelling in a description phase and generation phase. In the description phase the external description given by the designer is translated into either a set of rules and facts, which are processed in the generation phase with a greedy algorithm. The Semantic content generation framework [11] uses knowledge about entities in the world and attributes they possess. However top-down control alone is insufficient in cases where control over detailed and independent properties is desired. Search based approaches can offer both bottom-up and top-down control, respectively implemented with a search space and fitness functions. In [12] several search based approaches based on evolutionary strategies are investigated to find "good" and "diverse" maps. Abstractions of area control, exploration and balance are used as fitness function. With a process called novelty search, the diversity of levels is enforced. The diagram 1, slightly adapted from Togelius et al. [13], situates the search based approach with respect to generate-and-test and constructive approaches.

Most search based approaches rely on stochastic optimisation and indirectly imply a probabilistic model for the generation process. Some research tries to explicitly formalise content generators in a more general way as probabilistic models from which content instances can be sampled. The mathematical properties of such models allow for significant optimisations in the iteration process and provides in some cases finite time theoretical guarantees for the generator. For example Yeh et al. present a novel Markov chain Monte Carlo (MCMC) algorithm to synthesize constrained open world layouts, layouts where the number of objects are variable. They design a MCMC algorithm tailored to handle the change in dimensionality of the problem due to an unknown amount of variables. The probabilistic model which defines the search space is declared in an imperative programming language augmented with two probabilistic primitives, random variables and constraints.

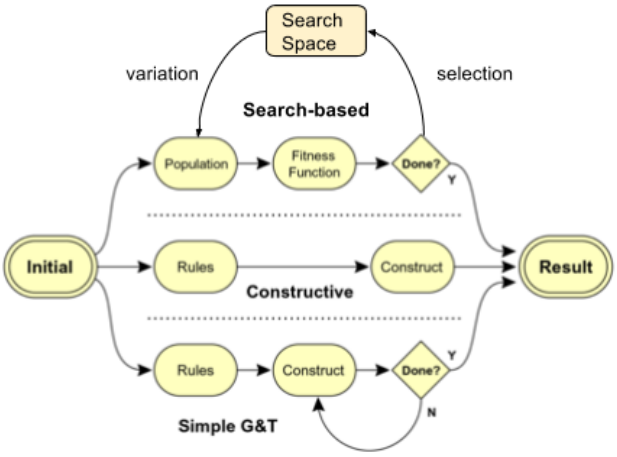


Fig. 1. PCG approaches

Learning based approaches can be used in order to further reduce the user workload using example data. According to Merrel et al. [14] the high-level requirements of building layouts, i.e. the form and placement of rooms and corridors, comes from complex considerations with respect to human comfort and social relationships. However, these considerations have resisted complete formalization. They try to tackle the lack of formalisation by learning them from data. The examples of high-level requirements in the data are extracted from an extensive catalogue of residential layouts.

B. *Solving design problems*

Content generation is not the only domain which tries to solve design problems. More specifically, I will discuss multi-objective (MO) design problems with possibly varying dimensionality which have been addressed using learning techniques. I focus on techniques that are relevant for the learning process in my methodology, and as such I will not address the limitations of their research but rather borrow its insights and apply them to the field of content generation. Karshenas et al. tackle the multi-objective search problem by estimating the search space density jointly with the objectives[15].They propose a multi-dimensional Bayesian network as the underlying probabilistic model for the joint distribution. Their motivation behind it, is to capture the dependencies between objectives and variables. Their experiments show significant results in comparison to state-of-the-art search algorithms based on evolutionary strategies.

III. Methodology

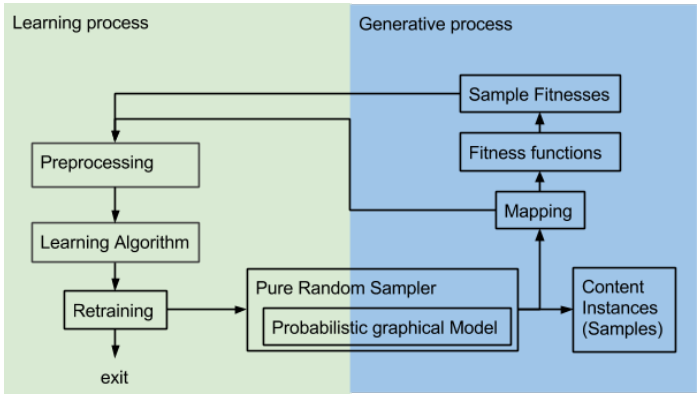


Fig. 2. Methodology overview

A generator with optimal expressive range would naively enumerate all possible varieties of a certain type of content, which results in a good expressive range, but poor performance. I want to maximise expressive range and still offer good performance. To do this, I optimise the search space, which is iterated in purely random way, of my procedural content generator. As mentioned in the introduction, a generator with both bottom-up and top-down control is desired. I define a new type of generator which offers both types of control, called the hyper-generator. It is made out of two process, (1) the generative process and (2) the learning process. In figure X is an overview of the main components in the hypergenerator.

The generative process has two main components, (1) the probabilistic model (bottom-up) and (2) the fitness functions (top-down) which can be defined by the user. In addition, the generative process requires a mapping which maps samples from the probabilistic model to a representation which can be evaluated by the fitness functions.

A. Generative process

The probabilistic model is a hierarchical structure where each node represents an asset type. For each asset type it is possible to define the range of its random variables. I focus on generating open-world layouts, therefore number of generated asset types is variable as well. The hierarchical structure is directed acyclic graph where each node only has a single parent and each parent can, as aforementioned, have variable amount of child nodes. The structure dictates the order for sampling the asset types, parent is sampled before child. The function which relates parent and child variables, called functional relation, can be used to declare an affine transformation between the variables.

In the hyper-generator the mapping is a simple one-on-one transformation of the samples into a representation for the fitness function to evaluate.

The sampled content is constrained with a set of fitness functions. These fitness function can be any injective function of the mapped representation to a real value. It informs the learning process which sampled layouts are desirable and which are not. I distinguish fitness function based on their type of hierarchical relation between instances; (1) pairwise between a parent and child node, (2) relative between siblings or (3) absolute over the complete layout.

In order to properly learn the search space distribution I need an unbiased set of samples, representative for the actual search space. The sampling is a recursive process, which starts with sampling the root parent node, followed by picking a number of children (n) from the user-defined range. Afterwards, n child nodes are sampled and the functional relation between parent and child is calculated.

B. Learning process

For the learning process I formulate the root learning problem of my approach in order to learn from the generation process. I want to learn two types of relations.

The first is between the variables of a parent and its child nodes, the second between the variables of sibling nodes, child nodes with a common parent. the aforementioned relations will be modelled as conditional probabilities, called conditional relations. The conditional relation models the relation between two nodes as the probability for the variables of a node given the variables of the other node. This fits in the generation process because the required order of first sampling the parent is not disturbed. The number of variables in my model can change due to the variable number of children. However, the probability of a random variable is defined by a probability density function, also called densities, which has a fixed dimensionality. To address this issue I will separate samples from the model into independent groups, each only containing samples with a certain amount of children.

To estimate the densities of the variables such that it has a higher probability for values which result in higher fitness I will use a learning method called supervised density estimation (SDE). Supervised learning, does not solely try to learn the structure of the data but requires additional labels for its data.

In contrast to the probabilistic model initialised in the generation process the densities are no longer uniform but defined by a Gaussian Mixture Model (GMM). I chose for a GMM because it is commonly used as model for the probability density of continuous measurements and it is cheap to calculate its affine transformation, necessary in the generation process, and its conditional and marginal distribution, both required for the learning process.

A single probability density function is not enough to learn the densities of variables in the generative process. However, training a density is an expensive operation. Therefore I propose two optimisation techniques to reduce the number of required training operations. The first, called variable sibling order, relies on the independence assumption between the variables of a child and its siblings past a certain order. The second, called marginalisation, assumes that the density which models the variables of a child can be marginalised from the density with higher dimensionality.

To estimate the parameters of a statistical model, a GMM in my case, I will use the most widely used parameter estimating method in machine learning called maximum likelihood estimation (MLE). The algorithm for finding the MLE I use is Expectation-maximization (EM). EM is commonly used for density estimation. However in order to use it for supervised density estimation, it needs additional methods. I will use two different methods. The first is called weighted density estimation (WDE), which estimates density given a dataset and their corresponding weights. These weights reflect the importance of the respecitve sample in the dataset[16]. The weights for WDE cannot be multi dimensional. The second technique I use, which does not suffer from this limitation, is called probabilistic regression. It estimates the density of layout variables V given their fitness F by calculating their conditional probability $(P(V|F))$ [17].

In order to use the learning algorithms, the samples from the generation process need to be transformed in proper training data with a fixed format. First, I extract the variables, which are part of the conditional density, from the hierarchical samples and format them in a flat vector. Secondly, I generate the fitness data, which are the labels for the SDE, by calculating the fitness values for all children of the samples and format them in a vector as well. After generating the training data I transform this data. First, I filter out samples with fitness values below a certain threshold. Afterwards, I apply a polynomial order to

the fitness values which affects the influence of fitness data on the learning process. The exact influence will be evaluated. Because the learning algorithm weighted density estimation only accepts a single weight, or label, per sample I need to reduce the dimensionality of the fitness vector to a single value, i define this type of dimensionality reduction as "single fitness dimension". I additionally add two other types of dimensionality reduction as parameters to be evaluated when using probabilistic regression, I define as "sample fitness dimension" and " sample fitness dimension".

The retraining component, repeats the training of the probabilistic model, with two different loops called retraining trials and iterations. For two reasons; first, to avoid sampling bias. Second, to increase the performance of the generator even further by repeating the training on an already trained probabilistic model. The sampling process is the sampling discussed in the generation with an additional ordering for generating the child samples. Because after updating the probabilistic model in the training process, each child is conditionally dependent on its previously sampled siblings. One additional problem when sampling a trained density is that its samples can exceed the user defined range of a variable. In order to respect the user defined ranges, a rejection sampling method is employed, which is is still purely random.

IV. Evaluation and discussion

Scene layouts in architectural context, which are a suitable type of content because it has both bottom-up and top-down properties. The constraints of the scene, defined as fitness functions, are inspired by the works of Yeh et al. and Merrell et al. An exhaustive analysis of the possible influence of each fitness function on the learning process is out of scope of this thesis. I will however evaluate the most commonly used constraints and compare them, based on their characteristics necessary to distinguish them in terms of behaviour relative to their input. I will evaluate the fitness functions minimum polygon overlap, target surface ratio, target distance, minimum centroid difference and closest side alignment.

Because expressiveness is a hard to measure quality [6], I will evaluate it with several different approaches based on visualisation of the density functions in the probabilistic model. However due the high dimensionality of the problem most of these require a more ad-hoc approach on low-dimensional use-cases (< 3). For one use-case I will do an exhaustive analysis of visualisations in two parts; (1) in terms of variance of the density and how much of the variance covers actually desired content and (2) in terms of bias towards particular values. In the first part I visualise the expressiveness in figure X, while considering the fitness function, first by comparing heat-maps between naive rejection sampling, also called generate and test, and rejection sampling with our method.

I visualise the density of the trained GMMs in two dimensions in figure X. This is done by draw their outer ellipses which contain the two order confidence level (95%) of its Gaussian components, called Gaussian surface area (GSA). Green is the true positive area, the area covered by both the trained model and the optimal search space, cyan the false positive area and red the false negative.

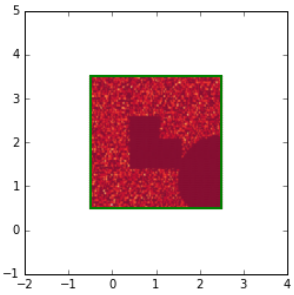


Fig. 3. Heat-map for naive sampling

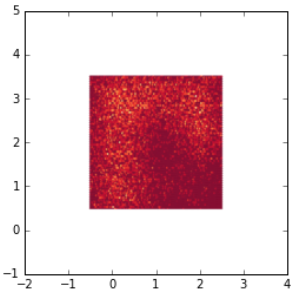


Fig. 4. Heat-map for sampling from trained model

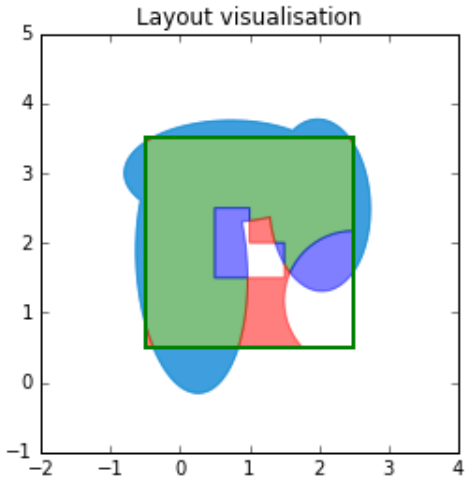


Fig. 5. GMM surface area coverage

The visualisation is used to calculate an analytical score for the expressiveness. The true positive area covers 89% and the true negative 85%. The second part visualises the expressiveness bias. I will do this by marginalising the variables from the joint distribution that models them.

On the figures with marginalised variables I show that the bias towards particular values is limited, the peaks in the trained density are close to the uniform distribution. To evaluate all methodology parameters I focus each time on one or two parameters, let them vary between a range, and analyse the results in order to be able to discuss their influence on the metrics. In these experiment expressiveness will be visualised by marginalising each trained variable separately, in terms of GSA. The performance of a trained model will be measured as the difference in average fitness before and after training, called average fitness gain (AFG). The computation time is measured as the amount of seconds to execute the learning method. I will average the results from these metrics over 5 probabilistic model each with a different fitness function, as mentioned in the beginning of this section. I summarise the evaluated parameters grouped according to their location in the methodology.

- Retraining: number of trials and iterations
- Data generation parameters: number of data samples, Pois-

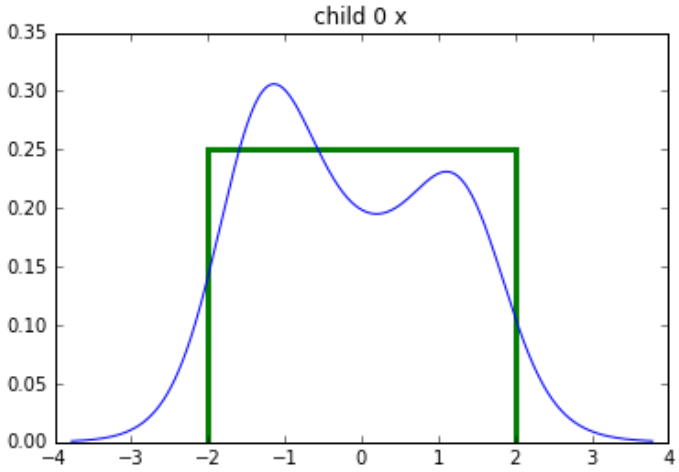
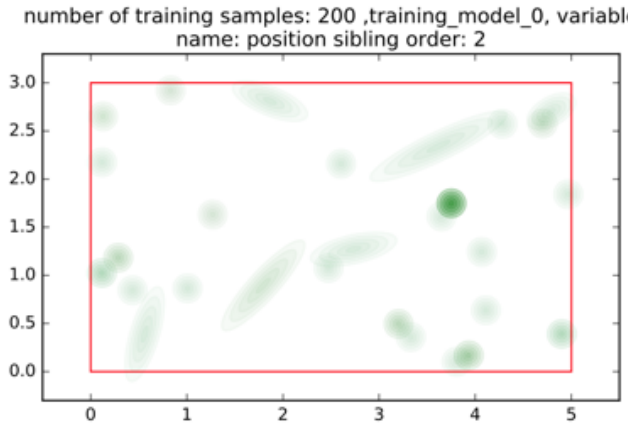


Fig. 6. Marginalised distribution for the x coordinate of child 0

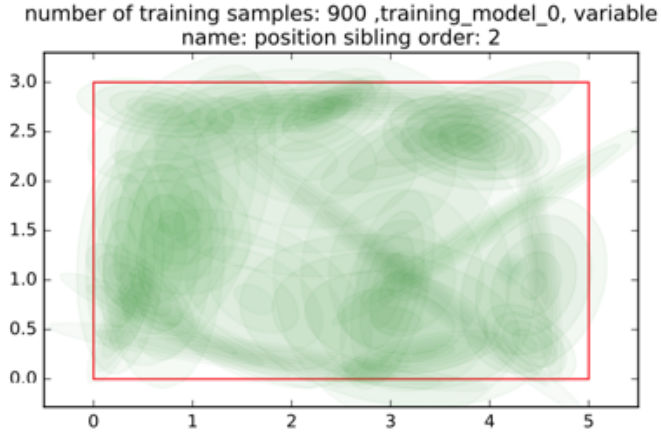
son disk sampling and fitness dimensionality reduction in probabilistic regression

- Fitness parameters: fitness polynomial order, fitness target, fitness hierarchical relation
- Hierarchical model parameters: maximum sibling order, GMM marginalisation
- Training model configuration: number of training variables

The last experiment is not a parameter of the methodology but evaluates the influence of removing one of the variable in probabilistic from the training data I will give the most significant results of these experiments and summarise how well each aforementioned fitness function has been optimised. Increasing the number of samples in the training data for the learning algorithm, significantly increases performance and expressive range. While only increasing computation time with 50% when quadrupling the number of samples. Probabilistic regression significantly outperforms weighted density estimation as supervised density estimation in terms of expressive range of the content, with similar results in terms of performance and computation time.



I was able to increase the average fitness value over all parameter configurations for the fitness functions target distance (22%), minimum polygon overlap (100%) and minimum centroid difference (45%). The other fitness functions; closest side alignment and target surface ratio, were harder to optimise and only reached respectively 5 and 10% improvement in the best of cases over all parameters.



V. Conclusion

I have presented a new approach which is able to optimise the search space of a content generator for some fitness functions. This is achieved by using a learning algorithm which automatically adapts a user-defined search space, with a minimal decrease in expressive range of the generated content. I have introduced a new metric for the expressive range of content generators based on the probability density function of the random variables in the search space. The probabilistic model in my methodology has two limitations; (1) it is restricted to continuous random variables, (2) its acyclic and directed structure cannot model cyclic generation process' like grammars or undirected generation process' like sampling from a factor graph. I think that my methodology, the hyper generator, of combining a probabilistic model, high level constraints and a learning algorithm which adapts the first to the lather, is sound and would be a strong alternative to other existing approaches for PCG. However, the results from current implementation are not significant enough to choose my method over standard search algorithms without adaptive search space, for example Markov chain Monte Carlo sampling. In order for the hyper generator to be a considered as alternative, I envision a more complex and suitable probabilistic model, which would be capable of capturing a wider range of fitness function with high accuracy. Therefore, I also suggest to drop any optimisations which reduce the model's complexity, in favour of reduced computation time.

References

[1] Newzoo, "Global games market will grow 9.4% to $ 91.5bn in 2015," 2013. Consulted on 25-May-2016 via http://www.newzoo.com/insights/global-games-market-will-grow-9-4-to-91-5bn-in-2015/.

[2] Y. Takatsuki, "Cost headache for game developers," 2007. Consulted on 25-May-2016 via http://news.bbc.co.uk/2/hi/business/7151961.stm.

[3] A. Iosup, "POGGI: Puzzle-based online games on grid infrastructures,"

Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 5704 LNCS, pp. 390–403, 2009.

[4] G. Kelly and H. McCabe, "A survey of procedural techniques for city generation," *ITB Journal*, vol. 14, pp. 87–130, 2006.

[5] G. Smith and J. Whitehead, "Analyzing the expressive range of a level generator," *Proceedings of the 2010 Workshop on Procedural Content Generation in Games PCGames 10*, 2010.

[6] N. Shaker, G. Smith, and G. Yannakakis, "Evaluating content generators," *Proced. Content Gener. Games A Textb. an Overv. Curr. Res.*, pp. 211–218, 2015.

[7] M. Schwarz, "Advanced Procedural Modeling of Architecture," 2015.

[8] P. B. Silva, P. Müller, R. Bidarra, and A. Coelho, "Node-Based Shape Grammar Representation and Editing," *Pcg*, pp. 1–8, 2013.

[9] K. Compton, B. Filstrup, and M. Mateas, "Tracery : Approachable Story Grammar Authoring for Casual Users," *Pap. from 2014 AIIDE Work. Intell. Narrat. Technol. (7th INT, 2014)*, pp. 64–67, 2014.

[10] D. Plemenos and G. Miaoulis, *Visual Complexity and Intelligent Computer Graphics Techniques Enhancements*. 2009.

[11] M. Bennett, "Semantic content generation framework for game worlds," in *2014 6th Int. Conf. Games Virtual Worlds Serious Appl. VS-GAMES 2014*, vol. 35, pp. 352–363, 2015.

[12] M. Preuss, A. Liapis, and J. Togelius, "Searching for good and diverse game levels," in *IEEE Conf. Comput. Intell. Games, CIG*, pp. 1–8, 2014.

[13] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-Based Procedural Content Generation: A Taxonomy and Survey," *IEEE Trans. Comput. Intell. AI Games*, vol. 3, no. 3, pp. 172–186, 2011.

[14] P. Merrell, E. Schkufza, and V. Koltun, "Computer-generated residential building layouts," *ACM Trans. Graph.*, vol. 29, no. 6, p. 1, 2010.

[15] H. Karshenas, R. Santana, C. Bielza, and P. Larranaga, "Multi-objective Estimation of Distribution Algorithm Based on Joint Modeling of Objectives and Variables," *IEEE Trans. Evol. Comput.*, no. c, pp. 1–1, 2013.

[16] "Pomegranate, probabilistic modelling library for Python."

[17] S. Ghahramani, *Fundamentals of Probability with stochastic process*. Pearson Education, 1994.

Contents

List of abbreviations

BO ..... Bayes Optimisation

SDE ..... Supervised Density Estimation

AFG ..... Average Fitness Gain

BIC ..... Bayesian Information Criterion

BU ..... Bottom-Up

CPPN ..... Compositional Pattern-Producing Networks

CS ..... Constraint Satisfaction

DMHD ..... Declarative Modeling by Hierarchical Decomposition

EA ..... Evolutionary Algorithm

EM ..... Expectation-Maximization

ESA ..... U.S. Entertainment Software Association

FPS ..... First Person Shooter

GMM ..... Gaussian Mixture Model

GSA ..... Gaussian Surface Area

GSA ..... Gaussian Surface Area

G& T ..... Generate-and-Test

MCMC ..... Markov Chain Monte Carlo

MIDEA ..... Multi-objective Mixture-based Iterated Density Estimation Algorithm

MIL ..... Multiple Instance Learning

MLE ..... Maximum Likelihood Estimation

MOA ..... Multi-Objective optimisation Algorithms

MO ..... Multi-Objective

NEAT ..... Neuro-Evolution of Augmenting Topologies

PCG ..... Procedural Content Generation

PDS ..... Poisson Disk Sampling

RTS ..... Real-Time S

SBPCG ..... Search-based Procedural Content Generation

TD ..... Top-Down

VOI ..... Variables Of Interest

WDE ..... Weighted Density Estimation

Chapter 1

Introduction

Computer games are no longer confined to the dark corners of arcade halls, where the stereotypical gaming teenagers passed their time. According to the U.S. Entertainment Software Association (ESA) four out of five households own a device to play video games and 42% of the American population plays 3 or more hours each week. [1]
On the current state of the gaming industry, New Media Consortiums stated in the 2014 K-12 Horizon Report: The [video game] industry is producing a steady stream of games that continue to expand their nature and impact they can be artistic, social, and collaborative, with many allowing massive numbers of people from all over the world to participate simultaneously. [2]
The global gaming market is one of the fastest growing markets in the world with a revenue of 83.6B $ in 2014 and a forecasted 91.5B $ for 2015 **??**. A similar trend is taking place in the Belgian gaming industry where the total revenue of the entire industry stood at 213 million in 2013 and is expected to generate revenues of $270 million in 2015. [3]
Suffice to say that the games industry is rapidly gaining in popularity and scale and that with this comes a demand for games with diverse content of varying artistic style and atmosphere for a more diverse target audience than ever before.
The chosen graphics, sounds, scenario and other types of content in games and the interaction with these components impact the player's experience. Enforcing coherence, in terms of design constraints, over all these components is not trivial.Focusing on global enforcement of these constraints can lead to conflicts that need to be resolved at the local level and vice versa. Their relation and how well these pieces of game content fit together are very important for a qualitative gaming experience [4].
As content quality becomes increasingly important to the economical success of a game, game developers are pushing their content production to unprecedented levels. However making large amounts of high-quality content by hand is becoming more and more infeasible. As its production is becoming too expensive [5] and unscalable [6], we are facing a content creation problem. GRiN, a recently bankrupted Flanders game developer studio, struggled with this problem. [7]
Therefore, (semi-)automatic means of generating content - as found in the academic domain of procedural content generation (PCG) - could help resolve the aforementioned content creation problem [8].
As content quality becomes increasingly important to the economical success of a game, game developers are pushing their content production to unprecedented levels. However making large amounts of high-quality content by hand is becoming more and more infeasible. As its production is becoming too expensive [5] and unscalable [6], we are facing a content creation problem. GRiN, a recently bankrupted Flanders game developer studio, struggled with this problem. [7]
Therefore, (semi-)automatic means of generating content - as found in the academic domain of procedural content generation (PCG) - could help resolve the aforementioned content creation

problem [8].
In theory, procedural content generators could be designed for any content domain. The uniqueness of games and the ongoing evolution of rendering techniques prevents us to be exhaustive but the supported content types include terrain models [9], level maps [10], music compositions [11], vegetation, architecture [12], weapons [13], character behaviour, stories [14] or quests [15]. Not only can generators replace or alleviate parts of manual authoring, they can also increase a game's replayability and unpredictability by presenting the user with a unique variety of the content in every gaming session. In literature, an often used metric for the contents variety is expressive range [16]. This metric should ideally measure the amount of unique and interesting content a generator can create.
To meet the demand of more unique and complex graphics, the techniques driving generators have increased in complexity over the years. Including heavily optimised meta-heuristics [17], grammar based approaches [12], formal logic [18] and machine learning algorithms [19] or even a hybrid approach of parameterized grammars and Markov chain Monte Carlo[20]. To facilitate generating large varieties of unique content the generator itself is often a pseudo-random process. This makes them hard to use for the people in charge of content creation, for example non-experts that are not specifically trained to understand the behaviour of the underlying techniques. The time and effort spent on learning the technical aspects on how to use these methods may defeat the purpose of using procedural modeling in the first place [21].
This results in a majority of the game industry, triple-A studio's in particular, still opting for solving the content creation problem by investing large amounts of resources in manual asset creation with artists [22] instead of using PCG [23].
Another consequence is that most PCG solutions applied in commercial projects rarely use techniques from academic sources and they are kept as simple and predictable as possible. Generators currently used by the industry exhibit three typical characteristics: (1) they are ad-hoc solutions (i.e. they only work for the specific game content they were designed for)[24] [25], (2) they are only used for non-critical content (i.e. failing to generate them correctly will not break the game), and (3) only techniques with a guaranteed correctness are used for critical content (i.e. the generation techniques have a finite set of possible content, that is verified to be always correct [26].

Current procedural generation approaches can be divided into two categories: constructive approaches and generate-and-test (G & T) approaches [27]. Constructive approaches are configured in a bottom-up manner: manipulating either low-level input parameters or the sequence of operations in the generation process. These methods generate content in a single step, i.e. one-shot generation, with no intermediate feedback to the user. This implies that the generator needs to guarantee a certain level of " acceptable" quality after a single execution. Achieving this often involves only performing certain sequences of operations or constraining the manipulation to certain input parameters as to guarantee to never produce "broken" or invalid content. An example of this approach is the use of fractals to generate terrains [9]. Constructive generators often use complex, recursive control logic. They require a bottom-up manipulation of the user. It allows direct control over the details of the procedure generating the content. This type of control is well suited for altering low level independent properties of the content, e.g. the color of the leaves in a tree. However, once started, the generative property forces the user to wait for it to end and evaluate the result. Consequently, technical artists often must tweak parameters and massage initial conditions, regenerating the content each time, to achieve a desired look [28].
Conversely, generate-and-test approaches generate content iteratively and search for the most fitting result across generations. G& T approaches are configured in a top-down manner to provide it with a higher level of control: manipulating constraint sets and fitness functions on the resulting content. For example in [29] they generate personalized procedural maps using evolutionary algorithms.
Generally speaking, the top-down manner defines what the content should look like while the bottom-up manner defines how it should be made. Iteratively generating instances and selecting the most fitting can be regarded as a top-down control over the procedure. This approach is

more intuitive as it operates solely on the final generated output and is agnostic of the specific manner in which the content is generated. This allows for much more general applicability than bottomup approaches. The user can enforce constraints over multiple properties simultaneously, e.g. adjacent leaves should have a slight different tint of green [30]. It can however require several iterations to produce content that has desirable properties depending on the complexity of the constraints in the test procedures and possibly even prove impossible if the constraints are in conflict with the generators initial configuration [31].
In the case of generating game content, characteristics of both types of control are desired [27]. The artist needs both direct and indirect control over a generator and the generated content to be able to produce qualitative and diverse content. As some content properties are inherently more suitable to realise with one of both types. An example suitable for bottom-up is direct control over visual artefacts in a game which the designer can create without any conflicts arising between them. Whereas a constraint which affects all game elements is more suitable to enforce using top-down control
In order to increase controllability, both types of control should be combined in such a way that its respective disadvantages are minimized, without hurting the control type's useful characteristics or reducing the content generators expressive range. My solution integrates bottom-up control by allowing the user to declare a search space, which defines the range of possibly generated content instances. As well as top-down control in the form of fitness functions which will guide the iterative selection process of instances from this search space. The most notable part is the automated optimisation of this search space, given the fitness functions, in order to select more content instances with "fit" properties. This reduces the number of required iterations to find the desired content and allows the use to declare the search space in less detail.

However, when optimising, it is important to minimize the number of "less fit" instances and reduce the search space, without hurting the diversity in the search space. The number of unique instances with desired properties in the search space, expressive range of the generator, could be reduced if the optimisation would carelessly exclude instances. That is why my solution will be evaluated both on performance of the iterative process and the expressive range of the result.

To achieve this my methodology consists of 2 main process', the first contains the *generation process* where I focus on generating content with maximum expressive range. The second process regards the generated content as data, and uses this data in a *learning process* to improve the aforementioned generation.

The second chapter elaborates on relevant domain knowledge from the PCG domain that is necessary to compare my generation process to other approaches in existing work both in terms of expressive range and controllability. And additionally work related to techniques used in the learning process. The third chapter is my methodology, which is similarly split in two parts. The first part explains the techniques used to generate the content. The second discusses the learning process and the manner in which it integrates with the generation process.

Chapter 2

Related Work

In the majority of previous work on procedural content generation (PCG), the problem of parameter tuning and controllability is repeatedly mentioned as an important track of future research. To control a content generator there are often a set of input parameters which can influence the content it generates. In PCG literature controllability has been referred to as how much a user understands the shift in resulting content relative to changes of the input parameters [32]. To evaluate controllability and procedural generation methods in general, the work by Smith et al. has suggested using expressive range. Expressive range refers to the space of potential content that the generator is capable of creating (including its bias towards generating particular kinds of content in that space)[16]. Expressive range is an important quality metric for PCG, especially for the games industry, because a generator with high expressiveness will have less "art fatigue". Art fatigue refers to the state of the consumer when the content or art being presented is too repetitive or uninteresting.

According to [16] there is a tradeoff between controllability and expressiveness. A larger expressive range can require the user to tweak more content parameters, a larger range of possible values for these parameters or both. However more elaborate input to control the generator means more time is needed to achieve the wanted results. Furthermore the larger range of values can make the generators behave less predictably. Another related tradeoff is between controllability and the desired detail in the content instances [33]. Increasing the detail of generated content may also lead to more input parameters needed to control the generation process. In turn, this increases the cost of content production, as more experimentation and expertise are required to produce desirable results [21].

2.1 Controllability in Procedural content generators

Most often content generators are made for a specific content domain including vegetation, buildings and cities. I will distinguish related work based on the techniques used rather than the type of content they produce, going into more detail on how each technique tries to solve the problem of controllability. This includes more intuitive input, improved user control, and automatic integration of results [34].

In the first section, the controllability of constructive approaches in PCG will be discussed. Next, I will discuss approaches that use visualisation and user-interaction to enhance controllability. The third section will describe declarative and semantic modelling approaches. In the fourth section, search based approaches of PCG are discussed. Afterwards the fifth section will

describe approaches that formalised content generators as probabilistic models. Finally, in the last section an overview is given of approaches that use (machine) learning methods on top of these probabilistic models. Learning is particularly well suited when a probabilistic model is available, because this allows to reuse the vast research of generative models in machine learning. Which will be discussed for content generation but also for different domains which try to tackle design problems that have one or more properties in common.

2.1.1 Constructive approaches

A complete overview of all constructive approaches is out of scope for this thesis. Because from the perspective of controllability most constructive approaches behave similarly, I will focus on one of the most common constructive approaches, namely generative grammars. Especially because they are also used in combination with other techniques discussed in other sections of this chapter.

Generative grammars is a type of rewriting system, that consists out of a set of valid symbols (that will be used to make strings) and a set of production rules (e.g. $a \rightarrow bc$ or $b \rightarrow da$) that will recursively expand the start symbol(s) into a larger string of symbols. The valid symbols can be divided into two sets: non-terminal (N) and terminal symbols (T). The left hand side of production rule can only contain non-terminal symbols where the right hand side can contain both non-terminal and terminal symbols ($N \cup T$).

From the start symbol, the production rules will be applied, replacing symbols from the left hand side with symbols from the right hand side until a terminal symbol is placed. Generative grammars typically operate on strings, but they are not restricted to that type of representation. Other types of grammars including, graph grammars capable of generating game missions as well as shape grammars to generate 3D buildings. Grammars can be used to generate many different types of structures, including graphs, tile maps, two or three dimensional meshes [35]. It is not necessary to let generative grammars run until the resulting string only contains terminal symbols. This implies that the grammar needs an ending condition to stop the application of production rules (for example limiting the total string length of the result).

A simple example would be two rules: $A \rightarrow AB, B \rightarrow A$. Which would result in the following strings after applying the rules for 3 iterations:
Start: A
Iteration 1: AB
Iteration 2: ABA
Iteration 3: $ABAAB$

In [35] they summarise the current types of game content described in literature which lend themselves well to generation with grammars, including vegetation [36], architectural structures (primarily buildings) and game levels. Plant modeling often relies on string replacement. In Francon et al. instead of using terminal symbols to decide when to stop the generation process, the evaluation is stopped based on the number of iterations and the produced string is subsequently mapped to a graphical object. In contrast, architectural modeling is usually inspired by shape replacement in a coarse to fine fashion (i.e. subdivision).

One of the most advanced grammars for buildings in the current literature is CGA++ [12]. In contrast with other grammar-based approaches CGA++ additionally allows context sensitive tasks like the merging of shapes to avoid overlapping geometries, for example to allow a single roof the cover a complete building.

A more general approach to modelling with grammars called General Grammar or G2 [37] adapts various concepts from the programming language Python. Their goal is to increase the descriptive power of the language without losing its semantics or creating ambiguity while the syntax

is kept human readable. More concretely, they allow for multiple types of non-terminal objects with domain-specific operators and attributes. In addition it is possible to define abstractions with structure templates and classes that have a set of declared parameters. This allows to combine several objects in the grammar, of which an example originating from Krecklau et al. is given below, in figure 2.1. Compared to strings, graphs are more useful to represent mis-

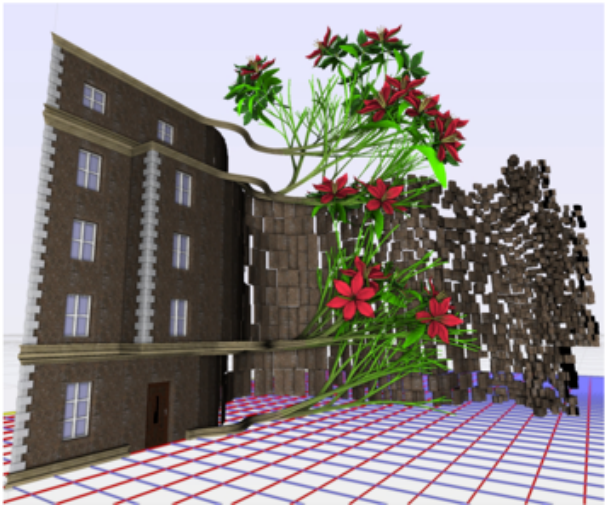


Figure 2.1: General Grammar example

sions (i.e. tasks to be executed by the player or other entities during the course of the level) and spatial structures for levels in games, especially when these need to have a certain level of sophistication [35]. Missions and spatial structures can rarely be represented as a linear or even tree like structure. A player can decide not to follow the main mission by taking one of the different possible extra mission, and later decide to return to the main mission. The generator has to guarantee that when a player does take different actions in the game, that the generated content does not hinder the player to finish the game properly. In [38] they combine graph grammars,as a means to generate mission, and shape grammars, that generate the spatial structure of the level, to produce complete game levels. The system generates the mission first and from this the shape grammar will generate the geometry of the game level.

Even though grammars are a powerful way to describe a numerous, or even infinite, amount of possible designs. Its formal, often textual, representation and recursive control logic is generally inadequate for artists. Which results in a steep learning curve. Other solutions try to avoid the text-based construction. Also, their fundamentally discrete representation cannot encode continuous variables without a loss of precision [39]. Another limitation is that each rule can only apply on one symbol or shape at the time. This means that once a certain replacement is applied it is no longer possible is no longer possible to query or merge back to it. For example, a grammar is enable to enforce symmetry between split-rules ($a \rightarrow bc, b \rightarrow d, c \rightarrow e$) [39].

2.1.2 Visualisation and user-interaction

Visualising the internal structure of a generator, such as a grammar, can help the user better understand the generators configuration or reveal the influence an input parameter has over the resulting content [40]. In this section I will discuss work which focusses on delivering a more intuitive control over grammars In order to facilitate the manipulation of grammars. The

real-time and visual editing system used in Lipp et al. [41] allows the user to construct a complete shape grammar. It adds an extension on shape grammars by implementing persistent and direct control over the instances produced by the grammar. This helps to add details to specific instances of content without having to rewrite the rules which affect all possible content. This is most notable when these details only occur once, which in a classic shape grammars would lead to complicated edge cases and bloating of the production rule set. This methodology illustrates clearly the advantage of mixing advanced procedural modelling techniques and standard design tools. In figures 2.22.3, originating from the aforementioned paper, illustrate how a user would add an edge case to a rule in a traditional grammar. The user would have to add new rules and exceptions manually(left). Whereas the new method, illustrated in the right figure, mitigates this tedious manipulation and automatically expands the grammar in an intuitive way in order to add the edge case without much effort. However if the user desires to add a larger amount of exceptions, optionally related to each other, he has to resort a lot of drag and drop movements. And repeat this step every time something changes in the iterative process of designing a building.

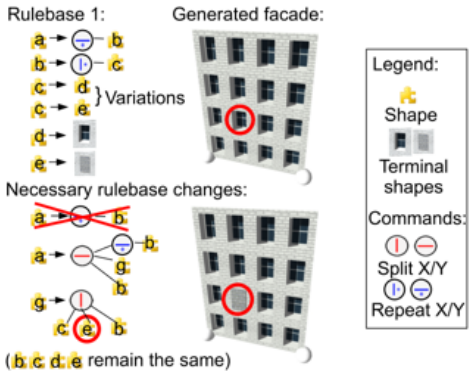


Figure 2.2: Edit facade by hand

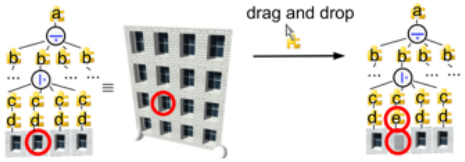


Figure 2.3: Edit facade with software

Silva et al. [42] introduce a node-based representation for a comprehensive design grammar by combining data flow from both the attributes and the shapes within cyclic graphs. Similar to the previous work they try to provide an artist with the tools to construct and manage a complex set of grammar rules for the generation of 3D. It is however a more generic approach which tries to tackle most of the limitations. Because each rule can only apply on one symbol or shape at the time it is no longer possible to query or merge back to it. In this paper they highlight the implications this has on the model's expressiveness as a lack of possible operations, including clustering, averaging, ordering or collectively managing a list or group of entities. To solve these limitations and keep the model compact and flexible their method introduces augmentations, encapsulation and attributes. Their argument is further strengthened by demonstrating examples that would not be possible, or impractical, to design with other grammar based approaches.

In figure 2.4 from Silva et al.[42] an example is shown of how a grammar can be visualised with connected nodes. Each numbered rule in the grammar corresponds to a node with the respectively matching number.

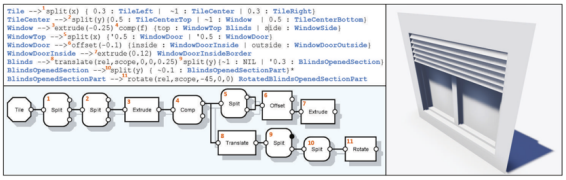


FIGURE 2: Window example modeled using a CGA shape grammar formulation (upper left; loosely based on [3] and [38]) and our procedural content graphs (lower left), each rule operation corresponding to a node with a matching number. The visual nature of our approach makes the flow of data easier to follow and does not require the user to come up with rule symbols (highlighted in blue) or geometric labels [6, 7], which generally make management difficult and error-prone.

Figure 2.4: Edit grammar using graph

Instead of visualising the grammar it also possible to increase user control by increasing the frequency of interaction by the user in the generation process. In Lipp et al. [41] they mention that in the majority of the generators the input is limited to tweaking of initial parameters and starting the procedure. Therefor they propose mixed-initiative PCG which requires more interaction with the user and reduces the amount of automation in the process. Thereby opening up a dialogue between the PCG process and the user. Yet another design system for grammars is Tracery [43], its goal is to enable novices to work with grammars that generate textual stories. Their approach is based on the principles of direct manipulation: "visibility of the object of interest; rapid, reversible, incremental actions; and replacement of complex command language syntax by direct manipulation of the object of interest" combined with iterative feedback by regenerating the story on edits. In the generated story a user can hover the mouse over a word that opens an inspector for a respective symbol in the grammar. More advanced user have the possibility to request a completely nested sequence of symbols, clearly showing each symbols hierarchy thanks to syntax highlighting.

Moving away from the grammars as underlying representation is necessary if the content in question is not suitable to for it. Tanagra [44] is a mixed-initiative design tool for 2D platform levels, like in Super Mario Bros, which allows to change its geometry indirectly and in real-time by editing the pacing of the level. The tool ensures that the level is always playable and its interface responsive by using a combination of constraint programming and reactive planning. Another type of levels that are intended for games with a top down view rather than the side view in platformers are called maps, often used in RTS or FPS games. Sentient Sketchbook [45] generates map sketches. Map sketches are a lower resolution version of the maps used in the full game which remove details not necessary for the core-gameplay. This abstraction helps to limit the amount of information displayed to the user, reducing user fatigue. Another reason for the abstraction is the underlying methodology, which is based on evolutionary computation. For the system to be responsive, it can only iterate up to 100 cycles which would be insufficient for larger and more detailed maps. For example in Togelius et al. [46] where they use evolution to generate Starcraft maps they need 100 000 iterations to find maps of a 100x100 resolution. The evolution is applied on an initial population seed by a user' s sketch. Because the user can directly input a complete estimate of the desired result, the system is capable of maintaining a certain degree of fidelity towards its initial design.

Sketchaworld [47], a procedural world generator, enables a non-specialist user to easily and efficiently create a complete 3D virtual world. Similar to previous tools it supports iterative modelling and direct manipulation of the content. In contrast to levels the complexity in creating 3D worlds does not arise from playability constraints but rather the integration of procedural techniques necessary to generate 3D models of both manmade and natural structures. Sketcha-

world is capable of fitting every feature between 3D models and their surroundings by using a semantic library. Which provides a high-level vocabulary for the various forms of interaction possible with the virtual world. Even though visualisation and user-interaction can speed up the design process and make it more intuitive, the user still needs to manipulate most details of the content by hand. For large amounts of content this becomes infeasible and a more top down approach is required in order to scale the configuration of a generator for all this content.

2.1.3 Declarative and semantic modelling

In order to achieve top-down control over the content declarative modelling tries to describe what will be computed instead of how it will be computed without specifying the control flow. This holds promise for avoiding accidental complexity. But by excluding how content should be generated exactly, the user loses the more detailed bottom up control which is defined in the generation procedure [18].

In [48], a book on intelligent computer graphic techniques, a whole chapter is dedicated to the use of declarative modelling for scenes. They divide the scene modelling in a description phase and generation phase. In the description phase the external description given by the designer is translated into either a set of rules and facts, called the internal description, that can be processed by the inference engine of an expert system, or a Constraint Satisfaction problem. The internal description is a set of arithmetic or geometric constraints on variables taking their values in their corresponding domains. Additionally in the description if the scenes become too large and overly complex, the user can apply declarative modeling by hierarchical decomposition (DMHD). This decomposes the scene into a set of sub-scenes with their own constraints and properties. In figure 2.5 an example scene decomposition is displayed, where the respective bounding boxes for each subscene are inside the bounding box of the parent scene.

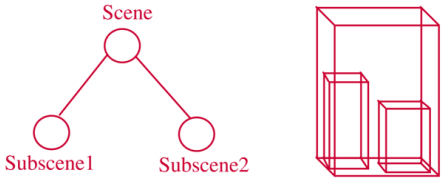


Fig. 2.1: The bounding boxes of the sub-scenes of a scene are inside the bounding box of the parent scene

Figure 2.5: Scene decomposition example

Next in the generation phase the internal description model, obtained in the description phase, is used to generate solutions during the generation phase. As the external descriptions made by a designer are generally imprecise in declarative scene modelling, more than one solution are often generated from a single internal description model. The generation phase often relies on an exhaustive enumeration of all possible solutions or a greedy algorithm. To reduce computation time of the relatively slow greedy algorithm they also provide some interesting generation optimisations using machine learning. One in particular is based on neural networks; given a few examples the hierarchical model learns the user' s preferences. This is materialized by a modification of the values of weights associated to the connections in the network.

Semantic modelling is a form of declarative modelling where all the possible declarations are researched with respect to a certain domain to facilitate the task for the user. The Semantic

content generation framework [49] makes use of knowledge about entities in the world and attributes they possess, as well as relationships they may posses with other entities. It builds upon established research results on parameterized procedural generation, constraint solving and semantic modeling, in order to automatically translate these semantic statements into a matching and consistent 3D virtual world. This provides a high level, semantic context for the generators of the different types of content in complex virtual worlds. However, not all constraints can be expressed semantically and may need to be managed at the model and logic level. And even though integration is addressed, during the generation process no interaction is possible which makes it more of a black-box type approach. In order to operate with a game engine and allow continuous operation throughout a game's lifetime a more sophisticated approach to generation is needed than the current method of instantiating the entire model in one pass [47][34].

The top-down control offered by semantic modelling helps the designer to enforce consistency over the content properties without having to worry about the details of each separate property. But in some cases it desirable to still be able to change these properties independently from each other without breaking the contents consistency. Achieving this would require an intuitive way of integrating, where necessary, the declarative model with details from the procedure which generates it. Preferably in a way that preserves the top down control of declarative modelling and the bottom-up control over the generation procedure.

2.1.4 Search based approaches

In order to find suitable content for approaches using declarative or semantic modelling, the techniques often rely on an iterative process. In search based approaches this iterative process is the subject being researched. By researching the details of the iterative process it could be possible to find methods which would result in less iterations. This approach offers both bottom-up and top-down control, respectively implemented with a search space and fitness functions.

Search-based PCG (SBPCG) is a more informed subtype of generate and test approaches. In SBPCG a search space is defined as the representation of the content that is used in the search algorithms. The content instances are tested using a fitness function, and assigned fitness values that correspond to their suitability for a specific game [27]. However in contrast to G& T, which relies on simple rejection sampling, SBPCG often uses stochastic optimisation, like genetic algorithms to find optima in the search space. When using grammars or declarative modelling the range of possible content is not defined explicitly in the procedure but rather a result of the consecutive decisions taken in the generation process. SBPCG offers top down control over the generated instances in the form of its fitness functions. In the domain of SBPCG however, they put more emphasis on the search algorithm, its performance and how to increase it. However, even though the search space of the content is explicitly defined, the details of the search process over this space will still have an influence over the resulting content which cannot explicitly deduced from the search process itself [50]. The diagram 2.6, slightly adapted from Togelius et al. [27], situates the search based approach with respect to generate-and-test and constructive approaches.

SBPCG components

Previous work has discussed SBPCG by defining 4 components [27] of which each has a distinct influence on the expressive range of the generator. This specifics of how the components influence the expressive range will be elaborated upon in their respective subsection. They use terminology from evolutionary computation but the concepts and considerations largely apply to other heuristic/stochastic search mechanisms. In previous work, it has also been suggested

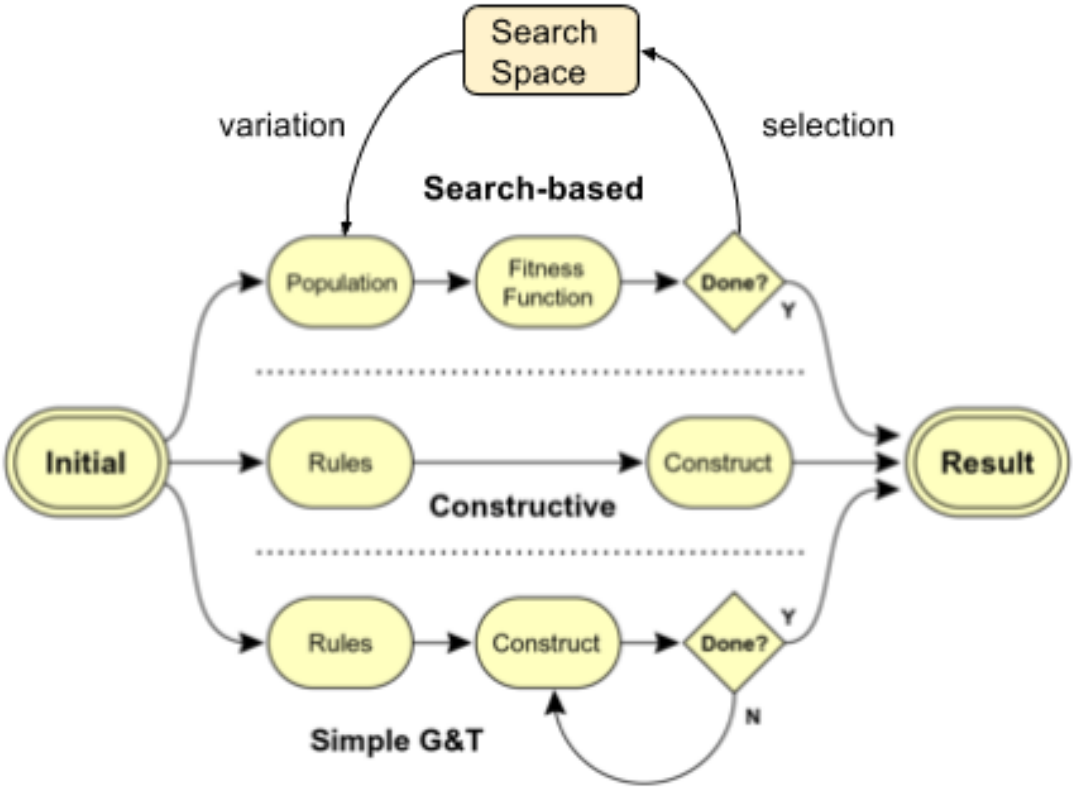


Figure 2.6: PCG approaches

that these components do not only apply to SBPCG but that a similar distinction can be made with respect to the constructive approach and G& T approaches .

The different components in SBPCG are to my knowledge never mentioned explicitly in research outside of procedural content generation for games. However, these components are straightforward to identify and compare between different domains. There has been research on analysing the influence of each component on the generator, most notably for the representation in level generator and their respective representative range [51][16][52]. My methodology is related but limited to the influence of the fitness function on the search space distribution, instead of analysing the influence ad-hoc I will leverage the result automatically to optimise the search space.

Firstly I will discuss the data structure which represents the search space, called the genotype. The second is the search algorithm. Once the search space is defined, the search algorithm, defines the way it is explored. The third component is the genotype-to-phenotype mapping. It maps the genotype to the data structure that is evaluated by the fitness function. Which is the fourth and last component.

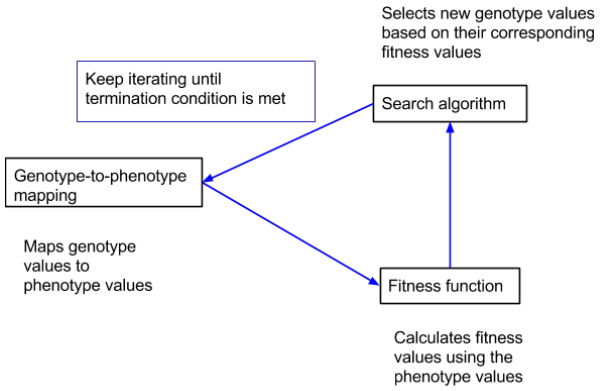


Figure 2.7: SBPCG components

Genotype When defining the content representation an important tradeoff has to be made between directly or indirectly encoding it. Similar to the map sketches in Sentient Sketchbook [45] it is possible to abstract away certain properties of the content when encoding it as a genotype. This allows to have a much smaller representation compared to direct encoding, where the size of the representation is linearly proportional to that of the phenotype. However, in indirect encoding a more complex genotype-to-phenotype mapping is required, both in terms of computation and implementation, and the abstraction can lead up to losing details in parts of the phenotype which are no longer encoded in the genotype [53]. Another implication of the representations encoding is the dimensionality of the search problem. When the representation is too compact it is no longer capable of representing all desired possible instances of the content or it can introduce undesired bias in the search space. However making the representation too elaborate introduces the curse of dimensionality into the search problem (see [54] for a review on representation within PCG context and [55] for a more general review).

Genotype-to-phenotype mapping Once the representation is chosen with the aforementioned considerations, there is an additional practical problem that can arise and should be avoided. Especially with indirect encodings, it happens that the representation can't be mapped to a valid phenotype. The consideration of indirect vs direct encoding is related to that of content generator expressiveness. An indirect representation could lead to a smaller number of variations in the content, if the properties which are measured are encoded by a short representation. For example a level encoded as a 10 character string versus a 5 character string. If for example expressiveness is taken very literal, one could measure in a scene layout the different possible positions of the furniture. Intuitively, it is possible to see that some layout will be very similar and should not each individually contribute to the expressiveness. That's why it is desired to incorporate a more qualitative metric like the "balance" of a scene, how well furniture is spread over the space, which is more relevant for the domain. Togelius et al. [27] addresses an interesting point in which the genotype-phenotype mapping can be seen as an PCG algorithm in itself. More specifically, how suitable this "inner " algorithm, the mapping, is as part of the "outer" generator, the SBPCG algorithm. The considerations made for a PCG algorithm in general will largely apply on the "inner" algorithm as well.

Search algorithm As previously mentioned, in the current state of the field, SBPCG is most often based on evolutionary computation. When researching papers beyond the generation of graphics specifically for the case of games, search techniques or optimisation techniques like Markov chain Monte Carlo (MCMC) methods [31] are more common. This does not mean that other search techniques like simulated annealing or particle swarm optimisation should be avoided. The most simple and unbiased way of sampling the search space would be uniformly. This is however very inefficient because in many cases a well defined search space expresses high locality. High locality means that a small change to the genotype should on average result in a small change to the phenotype and a small change to the fitness value. And it is required for an optimised search algorithm to actually perform better than random search. Optimised search algorithms will increase the rate of convergence but also exclude certain solutions to be found, and thus decrease the expressive range of the generator using these search algorithms. In the case of genetic algorithms the particular choice of mutation and crossover can result in accidentally ignoring certain areas of the search space [50].

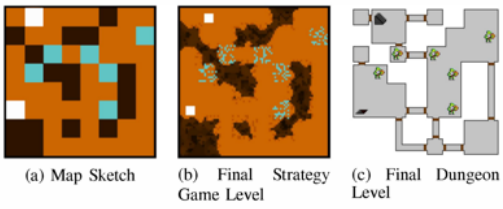
Fitness function After the search algorithm has selected an instance of the content, the fitness function will decide whether the instance is "suitable" to generate or not. This top-down control over the contents properties gives the fitness function the most direct influence on the generators expressive range. One issue with fitness functions is that they acts as a single objective, whereas content often has multiple properties that need to be enforced or optimised. It is possible to break the fitness function up in multiple fitness functions, but an often encountered problem with this approach is that it is not evident during the design phase whether they might conflict with each other [56]. This results in bad optimisation or complete lack of some of the desired properties. However, choosing a particular fitness function is a design problem in itself. It is ill-posed in the sense that the intent of a designer has to be translated into a function that can be evaluated, optimized and reflects exactly what he or she expects in the result. The intent of a designer is often not an objective metric but a subjective one. An example given in Togelius et al. [53] is that of a fitness function corresponding to content that is "fun, immersive, frustrating or exciting", which reflects a player' s affective state when he or she is experiencing a piece of content in a game. Up until this point in literature, no researcher has succeeded in completely formalizing this affective state.

Research on SBPCG applications

Now that the components and their respective considerations have been discussed it is easier to understand the challenges that arise in each generation problem that is solved using SBPCG. This section will overview related work on content generators for which the generation process partially or fully relies on search based approaches. Sorenson et al. [30] present a model that estimates a levels value of entertainment according to the presence of "rhythm groups", which map to switching moments of higher and lower challenge. The generative system represents a combination of genetic algorithms (GAs), for optimisation of the content, and constraint satisfaction (CS) methods, to fix broken individuals from the mutation process. The architecture provides a clear distinction between the level creation mechanism and its evaluation process. This results in a system that helps a human designer to "fill in the blanks", in this process both the rhythm groups and the user' s design are disrupted as infrequently as possible Shaker et al. suggest an approach to game level generation which strongly leverages the benefits and disadvantages of both the bottom-up and top-down approach, called the "progressive approach" [57]. This two-step approach starts with a search based technique, top-down, which is called grammatical evolution [58] that generates the level's timeline. The timeline describes the sequence of in-game interaction events that should occur at specific times during the game session. As playability is enforced on the timeline, generating the level using a constructive, bottom-up approach, from this timeline ensures the resulting level is playable. In their paper they highlight the advantage of their approach compared to other methods by highlighting that the playability check of timelines is much cheaper to compute than that of a complete level. In these other methods [59] the level is first completely generated and consecutively evaluated. Which often result in a significantly slower performance, because the more complex the playability demands are, the more computationally expensive the evaluation function becomes.

Game content is often a multidimensional data type and evaluating its fitness logically needs multiple fitness functions. Multi-objective optimisation algorithms (MOA) are specialised search algorithms that deal with these kinds of problems [60]. In [61] a multi-objective evolutionary algorithm (NSGA-ll multi-objective self-adaptive evolution strategy) is applied to generate levels for a real-time strategy(RTS) game called Planet Wars. The maps are evaluated on balance and dynamism which is required for a player to make the experience for the player challenging. From a game development perspective, on top of these gameplay constraints, the maps are required to be "aesthetic". This is achieved using several indirect metrics that retrieve the most attractive instances. The paper also contributes an additional level of abstraction over the evaluations metrics, where random forest classifiers are used to find the most significant metrics. Evolutionary computation is not only suited to find suboptimal solutions given a certain fitness, but it also very well suited to find a variety of solutions. In [62] several search based approaches based on evolutionary strategies are investigated to find "good and diverse" low resolution maps (in a grid of $16x16$). Abstractions of area control, exploration and balance are leveraged to construct metrics which can measure the "goodness" of a level. Using a process called novelty search, the diversity of levels is enforced as a fitness function to assist the evolution process in finding new and significantly different levels compared to previous instances, in other words increasing the generators expressive range. In figure 2.8 originating from Preus et al. which illustrates how the map sketch, which is used as genotype in the evolutionary algorithm, can be mapped to a phenotype of either a level for a strategy game or a dungeon.

In Raffe et al. [29] they try to combine several search algorithms and respective configuration to solve a common objective. In this paper they highlight that, besides evolutionary computation, there is another commonality between SBPCG methods: In contrast with their multiphased generation approach most SBPCG only use one evolutionary algorithm. Multiphased approaches have been applied for buildings in Tutenel et al. [63]. Another is Uriarte and Ontanón [64] which use multiple stages to generate RTS maps, each with distinct constraints. The paradigm all these solutions use is that of divide and conquer, by dividing the problem into smaller sub-



(a) Map Sketch (b) Final Strategy Game Level (c) Final Dungeon Level

Fig. 2: A map sketch (Fig. 2a) and the strategy game level it creates (Fig. 2b): white tiles are bases, cyan tiles are resources and dark tiles are impassable. The map sketch can also be envisioned as a dungeon (Fig. 2c), where white tiles are entrances or exits and cyan tiles are monsters.

Figure 2.8: A low resolution map and possible game levels

problems and trying to solve each subproblem separately, optionally with a different technique. Specifically for the case of SBPCG, they argument that if each stage can be independently optimised, these can all have their own search algorithm that runs in parallel. This strategy of specialised parallel evolution has been applied by Cook and Colton [65][66] to generate map geometry, player and non-player positions and a games complete mechanics simultaneously.

In some content generators a somewhat more involved form of evolutionary computations is used, called neuroevolution. In neuroevolution, artificial neural networks are trained through evolutionary algorithms, in which inspiration is found from the way biological brains evolved [67]. In neuro-evolution not only the state of the representation is evolved but also the structure of the representation itself. In [17] a Compositional Pattern-Producing Networks (CPPN) [68], is evolved through the Neuroevolution of Augmenting Topologies (NEAT) algorithm [69]. The network is a representation for the layout of objects with which the player can interact with during game-play, including enemies and pick-ups. In the game "PCG: Angry Bots", after a player has experienced a map, a recommender system is used to capture their feedback and construct a player model to evaluate future generations of CPPNs.

In an attempt to take PCG to the next level, Kerssemakers et al. [70] tried to implement a procedural procedural level generator generator. This generator would not only automatically generate content but a content generator in its own right. The system presented has two communicating generators iterating on different levels, an interactive genetic algorithm which acts as the outer generator which evaluates an inner generator. The domain researched is that of levels for Infinite Mario Bros. The inner generator is an agent-based algorithm which guarantees playability of the levels, and the outer loop let's a user decide which inner generator generates the most interesting levels. Originally, the procedural procedural level generator generator described in this paper was meant to be able to generate levels for any game for which levels could be described as a two-dimensional matrix (including other platformers, Roguelikes, Zelda-style, action-adventures etc). The constraints of the particular games would be included as parameters for the generators. However, this turned out to be a too ambitious goal at the time.

Even though some attempts have been mode to generalise SBPCG approaches, the current state-of-the-art is rarely able to generate content which is usable over more than a few specific games. Additionally, the configuration of the search algorithms requires a technical background, which excludes it use by artist without extensive knowledge on algorithms and data-structures. And even though the generation process can be fast, in order to achieve the exact desired results and reasonable performance the user will still have to manually tweak input parameters for some

time.

2.1.5 Probabilistic approaches

Most search based approaches rely on stochastic optimisation and indirectly imply a probabilistic model for the generation process. Some research tries to explicitly formalise content generators in a more general way as probabilistic models from which content instances can be sampled. The mathematical properties of such models allow for significant optimisations in the iteration process and provides in some cases finite time theoretical guarantees for the generator.

In favour of more general content generators Snodgrass proposes to explore probabilistic techniques [51]. They identified Markov chains (Markov 1971) as a viable approach to learning map structures. Markov chains are a method of modelling probabilistic transitions between different states. With relation to learning game maps, if the maps are generated as a set of tiles, then each different tile would correspond to a state, and we would learn the probability of generating a new tile given the previous one. This approach is promising because all maps contain some form of dependencies between local sections. He further expanded on the Markov chain technique by implementing a hierarchical Markov chain approach for Mario levels, using an abstraction technique. That is, after encoding the maps as a set of tiles, they further encode the utilized maps as a set of high-level tiles corresponding to different structures within the maps (e.g., pipes, slopes, platforms, etc.). Illustrated in figure 2.9 is a sample (direct) encoding for a section of Super Mario Bros originating from Snodgrass et al. However, the Markov-chains are only able

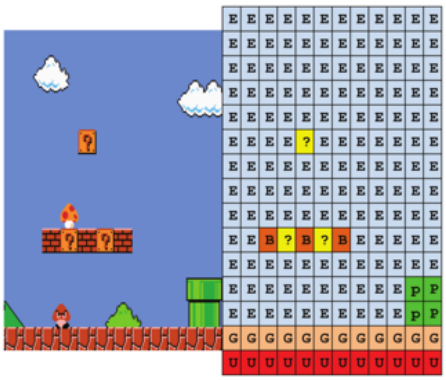


Figure 2: A sample encoding of a section from a *Super Mario Bros.* map. Color has been added for clarity.

Figure 2.9: A direct encoding of a Mario Bros level

to describe a discrete-step space, which excludes types of content which cannot be modelled discretely without loss of precision.

To synthesize constrained open world layouts, layouts where the number of objects are variable, Yeh et al. present a novel Markov chain Monte Carlo (MCMC) algorithm. The algorithm is called locally annealed reversible jump MCMC (LARJ-MCMC) and specifically tailored to handle the change in dimensionality of the problem due to an unknown amount of variables . The open universe probability distribution or search space is defined declaratively in an imperative programming language augmented with two probabilistic primitives, random variables and constraints. After declaration the constraints are encoded as factor graphs. A factor graph is a

bipartite graph that expresses how a global function of several variables factors into a product of local functions. These factor graphs are constructed dynamically once the number of objects is sampled. When sampling with MCMC from a distribution, a likelihood function is used to decide whether a certain step from the previous to the next sample is accepted. The likelihood is constructed from the factor graph and penalizes the new step if it distantiates the solution from the constraints [71]. While they can generate plausible and aesthetically pleasing furniture arrangements, this method does not completely synthesize scenes, since the underlying probabilistic program is written by hand and can only generate patterns that were expressed in the code [72]. See listing 2.10 for an example of pseudocode from Yeh et al. [71] to declare an open world layout. First the open world distributions are declared and afterwards the factors as constraints on the samples from the distribution. Another type of definition for open universe

(a)

```
def sample_plate():
    return Plate(
            position ~ (Uniform(-10, 10), Uniform(-10, 10))
            size ~ Uniform(0.1, 2.0))

def sample_table():
    return Table(
        num_plates ~ randint(1, 10)
        size ~ Uniform(15.0, 25.0))

Table = sample_table()

for i in 1:Table.num_plates:
    Plates[i] = sample_plate()

#Occupied area factor
Factor(SoftEq(0.7, total_area(Plates)/area(Table)))

#Inside factor
for Plate_i in Plates:
    Factor(SoftEq(0.0, area_outside_table(Plate_i)))

#Non-overlap factor
for (Plate_i, Plate_j) in pairs(Plates):
    Factor(SoftEq(0.0, overlap_area(Plate_i, Plate_j))
```

Figure 2.10: A declaration of an open world layout with constraints

distributions are stochastic or probabilistic grammars. In Talton et al. [39] they learn grammar rules for designs directly from data. The method used is grammar induction: given a set of design examples, they induce a probabilistic context-free grammar with the highest likelihood of generating said examples. Grammar induction can be used to quickly and easily generate a diverse set of new designs that have a similar style but are not identical to the originals. Their methodology is applicable to one common class of designs: those comprised of a hierarchy of labeled components. Such hierarchies are often found in creative domains, including architecture, geometric modeling, document layout, and Web design. Even though the stochastic context-free grammars can be a useful and compact generative representation, they have a few limitations mentioned previously in paragraph on procedural content graphs. In figure 2.11 is an example of japanese castles (bottom) which can be generated given a set of components (top right) and exemplar designs (top left) from the aforementioned paper [39]. In [72] they introduce a probabilistic model for scenes based on Bayesian networks and Gaussian mixtures which does not suffer from the limitation of discrete representation in grammars. However it is less general in its application, i.e. limited to furniture layout, due to its use of content specific knowledge. This knowledge describes contextual categories and originates from a learning algorithm (clustering). The algorithm groups objects occurring in a database of scenes according to the number of other objects they have in common which appear in the same scene as them. For example, spoon, fork and knife often appear in a scene in close proximity to both tables and a plates. These contextual categories allow the synthesis process to treat a wider variety of objects as

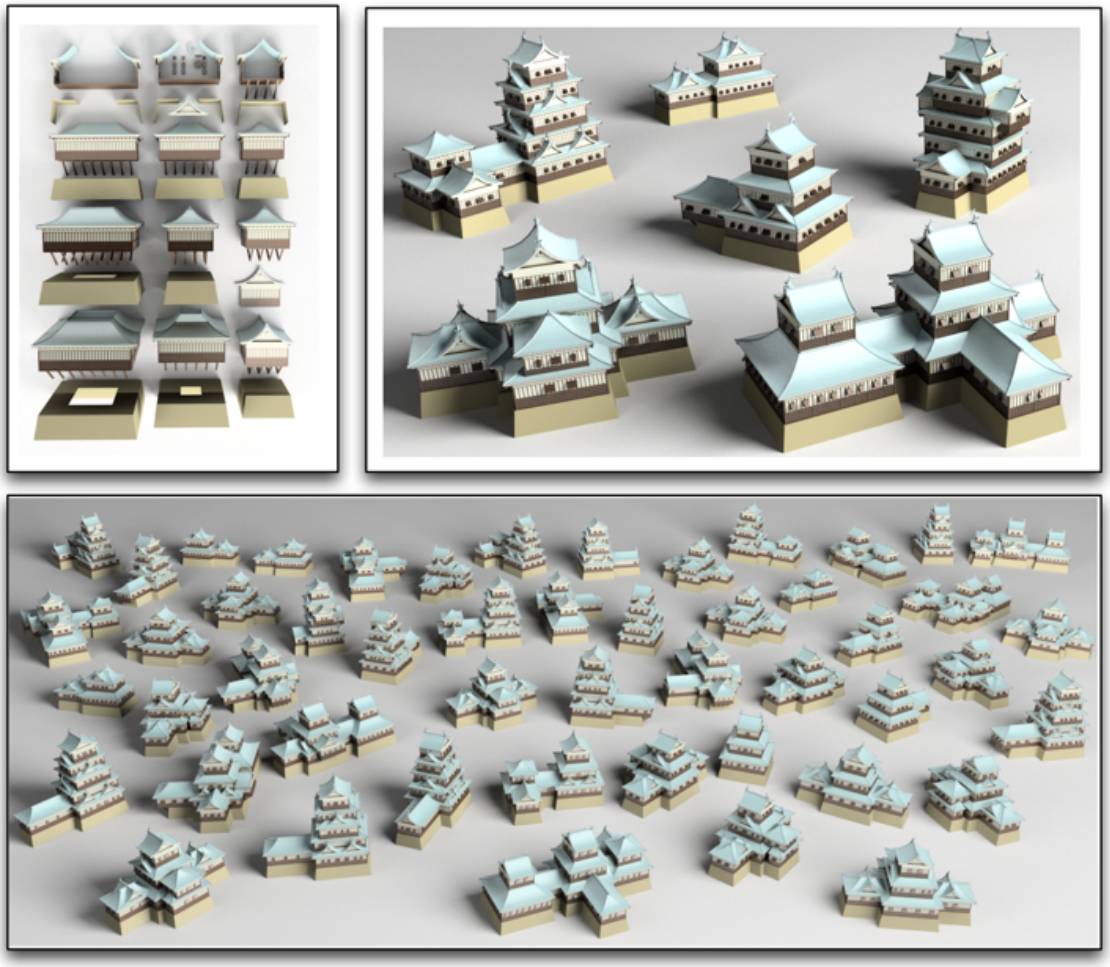


Figure 5. Japanese castles. (top left) The set of components. (top right) The exemplar designs. (bottom) Fifty distinct random derivations from the induced design pattern.

Figure 2.11: Generated Japanese castles

interchangeable. Once both the probabilistic model and the contextual categories are defined, the probabilistic model is trained on a controlled mix of user-provided examples and relevant scenes retrieved from the database. In the final step the trained model is sampled to generate contextually valid furniture layouts.

Another approach to furniture layout generation is based on established layout guidelines employed by practicing interior designers. In Merrel et al. [73] they argue that an effective furniture layout must address both functional and visual criteria. The functional criteria evaluate how well the layout supports human activities that naturally happen in a space, including conversation, rest, or movement. The visual criteria address the way a layout is perceived in its entirety. They formally describe some of these as a density function over scene variable properties. To interactively generate layout suggestions they rapidly sample these density functions by using a hardware-accelerated Monte Carlo sampler. Density functions are favoured because of the existing study of statistical distributions of human physical characteristics, such as body sizes and shapes, known as anthropometrics. This study establishes guidelines for the necessary clearance around objects and for the proper distances and angles between objects [74] In order to better understand the workflow of the system, the overview figure from Merrel et al. [73] is shown below. Given a user manipulation (i.e. moving a chair or rotating a table), the system suggests new arrangements which respect both user-specified constraints and design guidelines. This approach works well for realistic scenes but does not offer the freedom required to generate

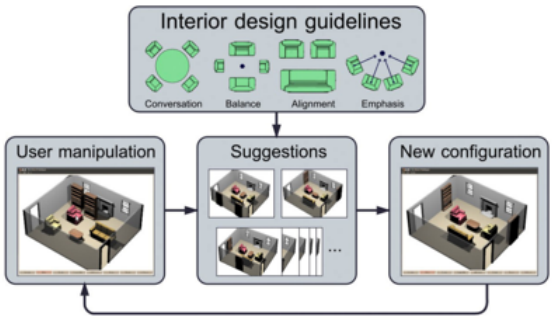


Figure 2: *System overview. In response to user manipulation, our system suggests new arrangements that respect user-specified constraints and follow interior design guidelines.*

Figure 2.12: Software work-flow for scene generation

scenes for games or movies where the scenes are designed with fantasy or science fiction theme in mind. These scenes often lack realistic constraints like established design guidelines or even our natural laws of physics.

The probabilistic approach offers a strong underlying mathematical model for the content, for which it is more intuitive to realise how the resulting content it will produce when sampled. Compare for example any grammar from the section 2.1.1, with text-based rewriting rules, and the more intuitively declared layout in Yeh et al. However, the probabilistic models still need to be manually designed by experts in order to generate professionally looking content.

2.1.6 Learning based approaches

Learning based approaches can be used in order to further reduce the user workload,this has been done in work from previous section 2.1.5 using example data [39][72]. However machine learning approaches are not limited to solely to learn from previously existing examples, the data can originate as well from simulations and physical models or even human feedback.

In a paper from 2003, Hertzmann [75] advocates the use of machine learning in graphics, especially Bayesian techniques. It gives a clear overview of the problems that could be mapped to computer graphic problems including density estimation, classification, regression, and reinforcement learning. Most notably are the many discussions in favour of leveraging uncertainty and probabilistic reasoning for learning the parameters of graphic model. Which means that learning can be used for tuning generator parameters in to order increase a generators controllability.

One such method employs the Bayesian technique of bringing in prior belief based on previous usage combined with expert knowledge. This is done in order to assist users in finding good parameter settings in as few steps as possible [76]. The problem of setting parameters for the procedural animation system is treated as one of optimization and is solved with a novel extension to Bayesian optimization. Bayes optimisation (BO) is a tool for the joint optimisation of sequential design choices of a black-box function. The evaluation of this function is expensive, therefore the amount of function queries need to be minimized. BO is capable of tuning the parameter values making up a particular design choice by estimating these values using only a few observations. Additionally the best location for the next query, which yields the most information about the unknown function, is estimated as well.

Another use of learning methods is to learn a generators objective function from data instead of designing it by hand. Carvalho et al. generate [77] game levels for an endless running game. This genre consists of an infinite race of which the player's goal is to run the longest distance possible while avoiding obstacles encountered in its path. The level is divided in playable game segments called chunks. In this work they focus on the chunk evaluation step, which employs neural networks that are capable of evaluating each chunk in terms of difficulty. The neural networks themselves were built on player data obtained from gameplay sessions.

Similar to the problem of learning valid furniture layouts for generation, is learning plausible object arrangements, useful in the domain of robotics. In Jiang et al. [78] they try to learn how objects relate to human poses, based on the pose requirements and the object's reachability. Their approach is to design appropriate density functions for the parameters of the 3D model by using a physical simulation of different human poses. Even though the approach is able to find correct object positions without the need to define any prior knowledge about their placement, the result lacks visual aesthetic appeal. Additionally, it strongly relies on the precision of the human model which in this paper only represents a simple generalised skeletal structure which cannot capture the many variations in human anatomy.

According to Merrel et al. [79] the visual aspect of building layouts, i.e. the form and placement of rooms and corridors, comes from complex considerations with respect to human comfort and social relationships. However, despite decades of architectural research, these considerations have resisted complete formalization. In the paper they try to tackle the problem of end-to-end automated generation of building layouts from high-level requirements. Due to the aforementioned lack of formalisation they opted for a data-driven technique. Their main focus is the generation of residential building layouts because these are less repetitive than the highly regular layouts often found in schools, hospitals, and office buildings. The data is comprised out

of a set of high-level requirements, such as the number of bedrooms, number of bathrooms, and approximate square footage which they extracted from an extensive catalogue of residential layouts. From this data an architectural program is synthesized, containing a list of rooms, their adjacencies, and their desired sizes. To learn the probability distribution over the space of architectural programs they define a Bayesian network which is trained on the aforementioned data. Using the bayesian network a novel architectural program is sampled. This sample only contains high level requirements and to generate an actual layout, in the form of floor plans, a stochastic optimisation method is used. In figure 2.13 from aforementioned paper [79] shows a table (top) with high level requirements extracted from real data with corresponding Bayesian network (bottom). One can wish to make content generation accessible to casual and novice designers who may lack a specific and professional intent or domain knowledge of advanced modelling techniques. In Talton et al. this is called exploratory modelling [80]. Their method is based on parametric design spaces, which are often high-dimensional and can vary widely in quality for different parameter values. The parameter values are used to as input for parameterized 3D models, in this case trees, humans and polygons. To find valid parameter configurations the system learns from the modelling history of a distributed community of users. These example models from previous usage are used to estimate the distribution of good models and their respective parameter configuration. Consequently, these estimates are used to drive more intuitive design tools with sliders and check-boxes instead of real valued parameters. The system is capable of active learning and will improve as more people use it. However, because of its elaborate effort to make 3D modelling more accessible, the tool no longer supports the direct control required for routine design tasks, such as the replication of a particular, precise model specification. Another limitation is that it is only usable for domains with good parametric representations.

The apparent advantage of reusing existing data is at the same time the weakness of the learning approach. In domains where data is expensive, sparse or even unavailable, these methods become unusable. Even if there is enough data, it is rarely in a format which can be used as is. In order to use it, a complex or laborious processing is required to convert the data to its desired format. Additionally most of these solutions treat the learning process as a black box which is too hard to control or understand which hampers the analysis of its influence on expressive range.

2.2 Solving design problems

Content generation is not the only domain which tries to solve design problems. More specifically, I will discuss multi-objective (MO) design problems with possibly varying dimensionality which have been addressed using learning techniques. This section focuses on techniques that are relevant for the learning process in my methodology, and as such I will not address the limitations of their research but rather borrow its insights and apply them to the field of content generation. First I will discuss the research domains whom try to solve design problems which have overlapping characteristics with PCG. Secondly, I will discuss some of the related work which applies learning algorithms to solve these problems.

2.2.1 Research domains

I will discuss research domains which rely on learning techniques to solve problems for which regular heuristics are not powerful enough. This could be due to the high dimensionality of the problem or an objective function which is expensive to calculate.

First I will discuss surrogate models, which is an engineering method used when an outcome of interest cannot be easily directly measured, therefore a (simplified) model of the outcome

Feature	Domain
Total Square Footage	$\mathbb{Z}$
Footprint	$\mathbb{Z} \times \mathbb{Z}$
Room	{bed, bath, . . . }
Per-room Area	$\mathbb{Z}$
Per-room Aspect Ratio	$\mathbb{Z} \times \mathbb{Z}$
Room to Room Adjacency	{true, false}
Room to Room Adjacency Type	{open-wall, door}

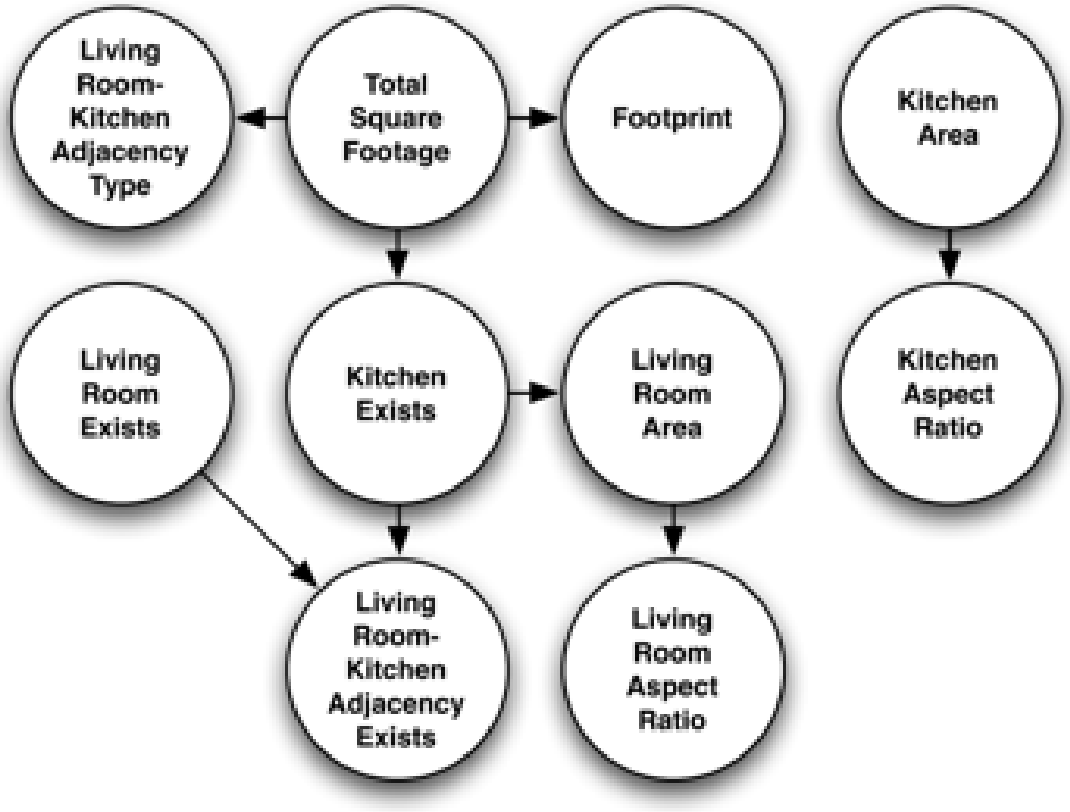


Figure 2.13: Generic floor layout requirements and example Bayesian network

is used instead. Most engineering design problems require experiments and/or simulations to evaluate a design objective with constraints. For example, in order to find the optimal airfoil shape for an aircraft wing, an engineer simulates the airflow around the wing for different shape variables (length, curvature, material, ..) [81]. This has been successfully applied in Kurek et al. [82] where a novel technique is presented that uses meta-heuristics and machine learning to automate the optimization of design parameters for reconfigurable designs.

Another domain for which design problems are solved by learning is hyper-heuristics. Hyper heuristics are a search method or learning mechanism for selecting or generating heuristics to solve computational search problems. The distinguishing feature of hyper-heuristics, compared with standard heuristics, is that they operate on a search space of heuristics (or heuristic components) rather than directly on the search space of solutions to the underlying problem that is being addressed.Two main hyper-heuristic categories can be considered: heuristic selection and heuristic generation. I will focus on the category of heuristic generation. More specifically hyper-heuristics which improve an underlying heuristic with learning techniques. Which is used for example in the optimisation of iterative compiler techniques where they try to automatically focus any search on those areas likely to give the greatest performance [83]. The method of learning from a content generators and leveraging the learned models has been proposed in [48] explicitly to increase convergence rate.

An interesting path to augment PCG solutions, which use iterative process', would be to solve the problem of content generation with hyper-heuristics. In order to relieve some of the effort of optimising the iterative process from the user.

2.2.2 Learning applications

The issue of maintaining expressiveness in content generation can be compared to the exploration-vs-exploitation tradeoff in an exploration problem. In an exploration problem the search space is not completely known, an agent exploring this search space has to balance the tradeoff between finding optimal solutions, exploitation, and increasing its information over the search space, exploration. The exploration process does not search for the best solutions but for diverse solution, preferably maximizing the difference with previously obtained solutions.

In Calinon et al. [84] they consider the exploration of the search space as a density estimation problem [84]. This approach yields additional information, such as shape and curvature of local maxima which can be used both to optimise exploitation and guide the exploration. They show that the search space can be generalised to a Gaussian Mixture Model (GMM) with an adaptive number of components. It is a black box approach capable of handling optima which changes over time. This is illustrated with an agent playing a dart game. The solution space of this problem is characterized by a single global optimum at different locations on the dartboard. The agent searches for the position on the board which maximizes the score, while progressively improving its throwing capabilities. Figure 2.14 originating from Calinon et al., they show the evolution of the search process. With the colored heat maps representing the expected distribution for the current throw accuracy of the agent (sigma in the figure). While the white ellipses represent the GMM for the promising targets, discovered by the agent. However, the control parameters of the density estimation have to be set by hand.

An example in which both surrogate modelling and hyper-heuristics are combined is a paper of Feliot et al. [85], in which they propose an adaptive Markov chain Monte Carlo (MCMC) sampler using Bayes optimisation. Bayes optimisation has two main components, the first is a probabilistic surrogate model which captures our prior knowledge or beliefs about the behaviour of the unknown function, in this case the performance of the MCMC sampler. The second component is a loss function that describes how optimal a sequence of function evaluations are,

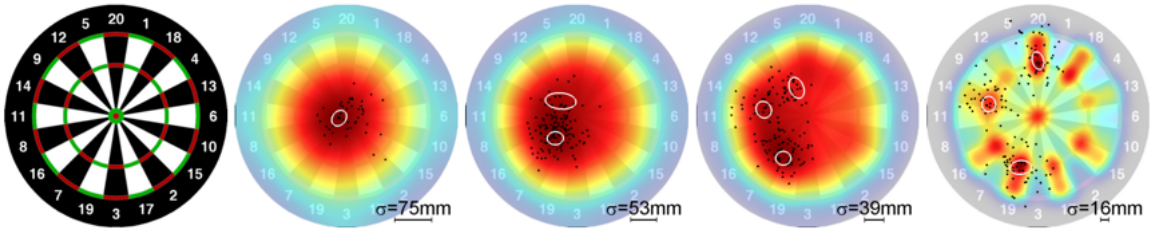


Fig. 1. Evolution of the search process. The colored heatmaps (corresponding to steps 1, 1060, 1757 and 3000) represent the expected distribution for the current throws accuracy of the agent, with colors from cyan to dark red linearly spread between lowest and highest scores. Steps 1060 and 1757 are the steps when Gaussian splits occurred. The black dots show the last $L = 150$ trials. The white ellipses represent the mixture of Gaussians representing the promising options discovered by the agent (approximation of the unknown solution landscape by iterative exploration and adaptation).

Figure 2.14: A dart game played by an agent evolving over consecutive throws

this component is not relevant to the discussion and further elaboration is omitted. Feliot et al. propose an adaptive strategy consisting of two phases: adaptation and sampling. In the adaptation phase Bayesian optimization is used to generate a sampling policy. In the sampling phase, the generated policy is used to adapt the MCMC sampler and consecutively sampled from. This methodology mitigates the need for manual tuning of the MCMC parameters in order to achieve state-of-the-art performance, Bayesian-optimized MCMC is able to realize the same gains without any human intervention. However scaling to higher-dimensions remains a challenge.

Karshenas et al. propose a broader approach to density estimation [86] where the multi-objective problem is tackled by not only estimating the search space density but the objectives as well. But instead of modelling everything in a single level, without hierarchy, they propose a multi-dimensional Bayesian network as the underlying probabilistic model for the joint distribution of search space variables and objectives. Their motivation behind it, is to capture the dependencies between objectives and variables. In addition to approximating the multi-objective solutions, or Pareto-set, they are also able to estimate the structure of the problem search space, which enables to decompose it in smaller sub-search spaces. Their experiments on several multi-objective problems with different objective space dimensions have shown significant results. In comparison to state-of-the-art evolutionary algorithms, with a search based on conventional genetic operators, the algorithm performs significantly better on some and comparable on other problems. Figure 2.15 from Karshenas et al. shows an example of the structure of a multi-dimensional Bayesian network for classification. In this type of model, the nodes are organized in two separate layers: the top layer comprises class variables and the bottom layer contains feature variables. The arcs can be partitioned into three subsets. Which results in a subgraph for the directed relation between class and feature nodes, a subgraph for the class nodes and their relations and one for the feature variables and the interactions between them.

An important aspect in MO problems is the preservation of diversity when exploring the Pareto-set. The preservation of diversity can be compared to that desire for expressiveness when generating content. Bosman et al. [87] use mixture distributions as an underlying probabilistic model in their method. It models the probability distribution of a selection of the available solutions while optimising a certain problem with an evolutionary algorithm (EA). Their general algorithm is called Multi-objective Mixture-based Iterated Density Estimation Algorithm (MIDEA). Listing 2.1 illustrates how the learning technique is integrated with the EA.

Listing 2.1: MIDEA pseudo code

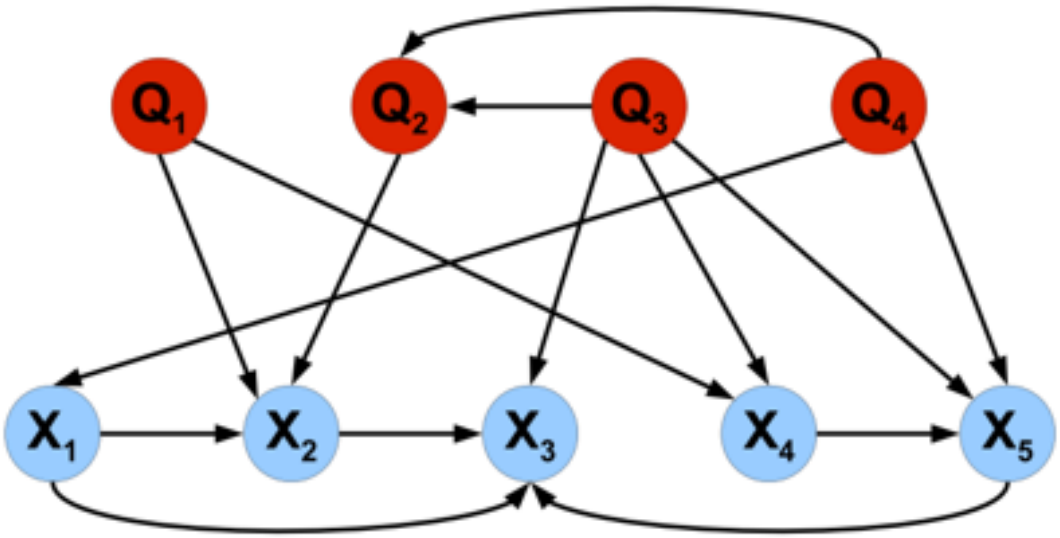


Fig. 1. An example of a multi-dimensional Bayesian network structure.

Figure 2.15: A multidimensional network Bayesian network structure

```
1 Initialise a population of n random solutions and evaluate their
   objectives
2 Iterate until termination
        2.1 compute the domination counts
        2.2 Select solutions with the diversity preserving selection
           operator
        2.3 Search for a (mixture structure) M
        2.4 Replace non-selected solutions with new solutions drawn
           from M
        2.5 Evaluate the objectives of the new solutions
```

By using mixture distributions in EA's, the distribution is able to process complicated nonlinear interaction between problem variables. For which an example is given in figure 2.16 from [87], where a non linear dependency in the sample (left) is approximated by two normal density functions after density estimation (right). They show that compared to state-of-the-art multi-objective evolutionary algorithms their algorithm has comparable optimisation results on a set of sixteen MO problems. Furthermore, MIDEA's display unmatched behaviour in obtaining diversity and a good distribution of trade-off, especially for higher dimensions.

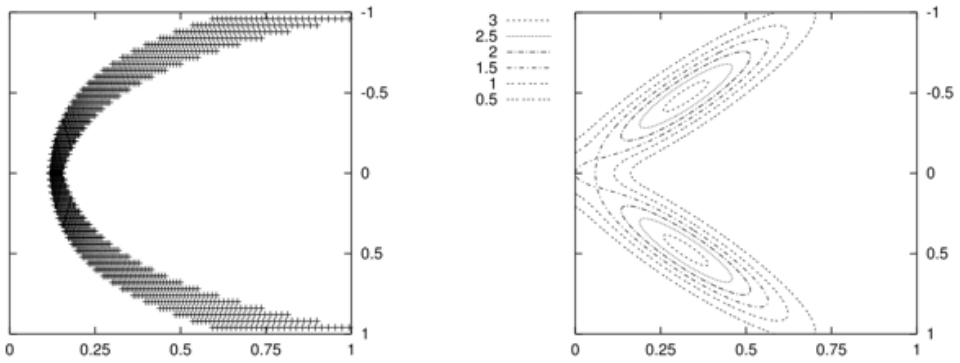


Fig. 2. A non–linear dependency in the sample set (left) and the contour lines of the estimated probability distribution using two normal pdfs after clustering (right).

Figure 2.16: A multidimensional network Bayesian network structure

Chapter 3

Methodology

A generator with optimal expressive range would naively enumerate all possible varieties of a certain type of content. A significant portion of these varieties are unusable, which makes this solution both computationally intractable and uncontrollable. Suffice to say that a naive way of generating content iteratively, results in a good expressive range, but would result in poor performance. A lot of research tries to increase performance by designing an optimised heuristic, which no longer enumerates all possible varieties. In order to maximise expressive range, I will not optimise the iterative process but instead optimise the search space of the procedural content generator by adapting the probability from which the varieties are sampled.

From the extensive literature study on controllability in PCG it is apparent that a generator can offer control to the user which is top-down, bottom-up or both. In the works with both types of control the integration and coordination between the distinct types was often an issue [47] [80] [59]. Which often resulted in one of both types being more present in the methodology, while the other type would be less elaborated upon.

As aforementioned in the previous chapter, constructive and generate-and-test approaches offer each only a single type of control, bottom-up and top-down respectively. Which is why I want to combine both approaches in a single generator. Ideally the combination of both approaches, combining the advantages of both a bottom-up and top-down perspective while negating its downsides, would yield a system that delivers optimally controllable content generation. These advantages and downside, mentioned in the related work, are each other's opposite depending on the perspective. In summary bottom-up control offers fine grained control over independent variables of the property, necessary to create detail. Top-down control offer high-level control in order to enforce a coherency among multiple assets, without the need to tweak every asset variable separately. I want to give the user full control over each parameter separately, but also the possibility to jointly constrain them.

My methodology of combining bottom-up, top-down and learning can be compared to a general example of human habits. After years of experience people create a set of habits, which they execute without much thought, they "know" their influence if they are consistently repeated. These habits can be viewed as rules we apply on our lives. However, over time, a person can notice undesired changes due to these rules. The person will learn from these changes, and potentially change the habit that caused it. significant results. In comparison to state-of-the-art evolutionary algorithms

To support both types of control simultaneously in a flexible way, I define a new type of procedural generator called a "hyper-generator", originating from the term "hyper-heuristic" (a search or learning method which optimizes a heuristic). The hyper-generator is made out of two

process' , (1) the generative process and (2) the learning process.

In this chapter I will discuss both the generative and learning process by breaking them down in their respective components. I start with an overview section which relates each component on the level of the learning and generative process. Additionally, I briefly discuss the components with some examples. Consecutive to the overview I will elaborate on the generative process.

The generative process has two main components, (1) the probabilistic model (bottom-up) and (2) the fitness functions (top-down) which are defined by the user. In addition, the generative process requires a mapping which maps a set of values sampled from the probabilistic model to a representation evaluated by the fitness function. The sampled values of the probabilistic model are not the content which will appear in a game. They are the values for the layout of the game. The actual generation of assets in a game, given a certain layout, is a different generation problem called asset generation. For example a scene of a restaurant with different types of furniture. The positions of the furniture is the layout, the form and color of the furniture are the assets. I will only address layout generation in my thesis. Finally I will elaborate upon the sampling itself.

In the section of the learning process, I will first elaborate on the specific learning problem I will solve. Secondly I discuss the components of the learning process. The probabilistic model, the learning algorithm, the pre-processing of generated samples, and the retraining in the learning process. In the third section I elaborate on an adaptation of the sampling discussed in the generation process.

3.1 Overview

In figure 3.1 is an overview of the main components in the hyper-generator. The generation process has two control mechanisms: a probabilistic model and fitness function. Respectively offering bottom-up and top-down control to the user. Firstly, the probabilistic model declares a valid interval, in the form of a uniform distribution, for each variable of all asset types. From these intervals, a variable space is constructed called the search space. The generation procedure samples from this search space to create specific asset instances (i.e. content).

Secondly, the fitness functions are a set of constraints on the variables which will evaluate the content (solid line arrow). These constraints reflect how "fitting" the content with certain variables are. The fitness of an asset instance in a layout is defined in relation to the variable values of other instances. Because the independent fitness of an asset instance can be optimised separately for each asset. But separate fitnesses do not control multiple assets simultaneously, in contrast to top down control over the content.

In the learning process, the fitness values from evaluation the fitness functions are used to constrain the search space of the probabilistic model, such that in a consecutive round of sampling the search space will result in content with on average higher fitness values. Using these fitness values, the learning process trains a probabilistic model (dotted line arrow), as such estimating a new probability density function for the variables. This means that values of variables which result in higher fitness for the content have a higher probability in the density function. The retraining component will decide whether this process needs to be repeated.

From the perspective of machine learning the probabilistic model acts as the "a priori" knowledge the user has over the possible content. Similarly, in a learning context, the fitness function values are regarded as "a posteriori" knowledge over the search space of the content. "A priori" refers to the information that can be deduced from knowledge available before observation of a process. Once a process has been executed one or more times it is possible to use the resulting

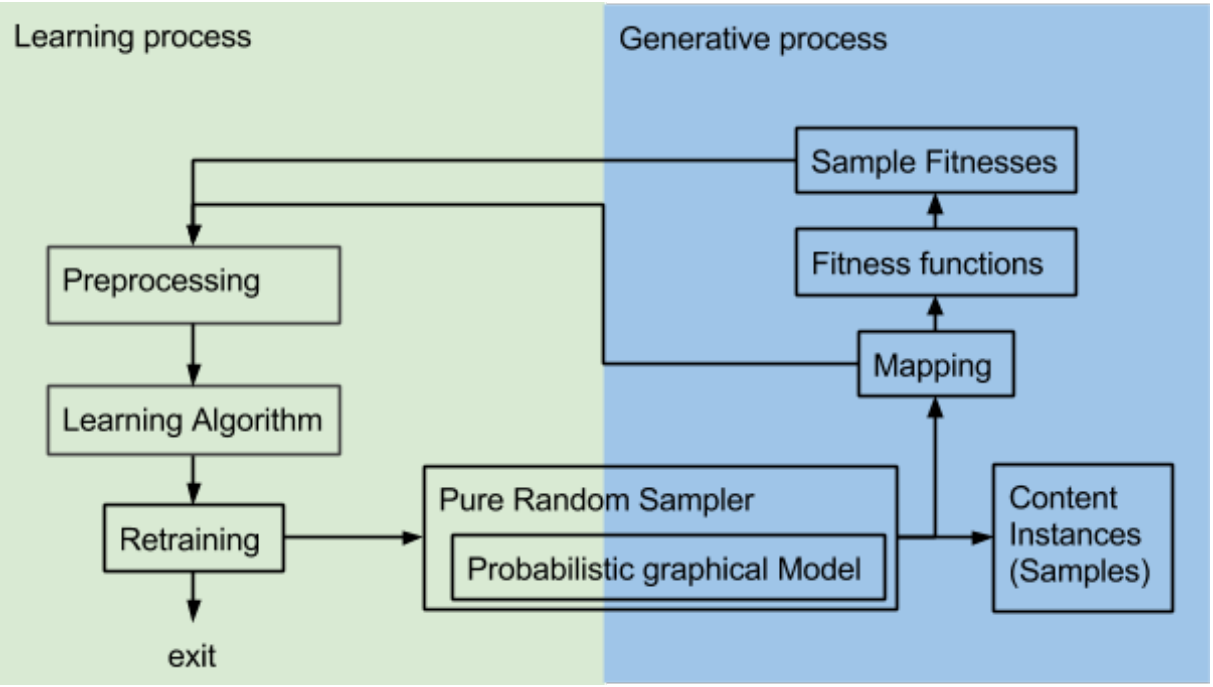


Figure 3.1: Methodology overview

observation to deduce "a posteriori" knowledge.

For example, in the probabilistic model I could define variables for a wall of a room with the analytic equivalent of the semantic declaration "bricks piled in a specific way, without gaps and with a size between 20 and 40 centimeters". Where-as a semantic top-down fitness function could be "walls that enclose a room with a surface of 20 square meters". The user does not directly state the room' s shape, it is up to the learning process to find it. This example also illustrates that one approach can be favoured over the other to define content on different levels, "size of bricks" and "surface of a room". In the hyper-generator the user is free to choose on which level each variable is defined for the resulting content, whichever makes most sense to them. However, it is not always clear which definition is most suitable for a particular content type. Referring back to the example of a wall in a room. A user could wish define the wall partly in a bottom-up and partly top-down way, respectively in terms of brick size and the rooms surface. When using both definitions there arises an uncertainty, "Where, within its interval, lie the best values for a variable according to the fitness". For example it could be that the distribution of brick sizes for a surface of 20 meter squared is not evenly distributed but rather favours bricks of size 20-25 and 30-35. If I can learn the approximate distribution of brick sizes, as a probabilistic model of it, that will more often result in rooms of 20 square meters, than our search algorithm, which samples from these brick size distributions to generate walls, will require less iterations. Figure 3.2 visualise a concrete definition of a probabilistic model and fitness functions. As aforementioned in the related work, a hyper-heuristic can use a learning technique which adapts an underlying heuristic. The hyper-generator will automatically optimise the search space of the underlying G& T generator, hence its name. The search space is optimised by the learning process as described in previous paragraph. Conceptually, a G& T generator can be regarded as a heuristic. As both the heuristic and the G& T generator search for suboptimal solutions by iterating over a search space.

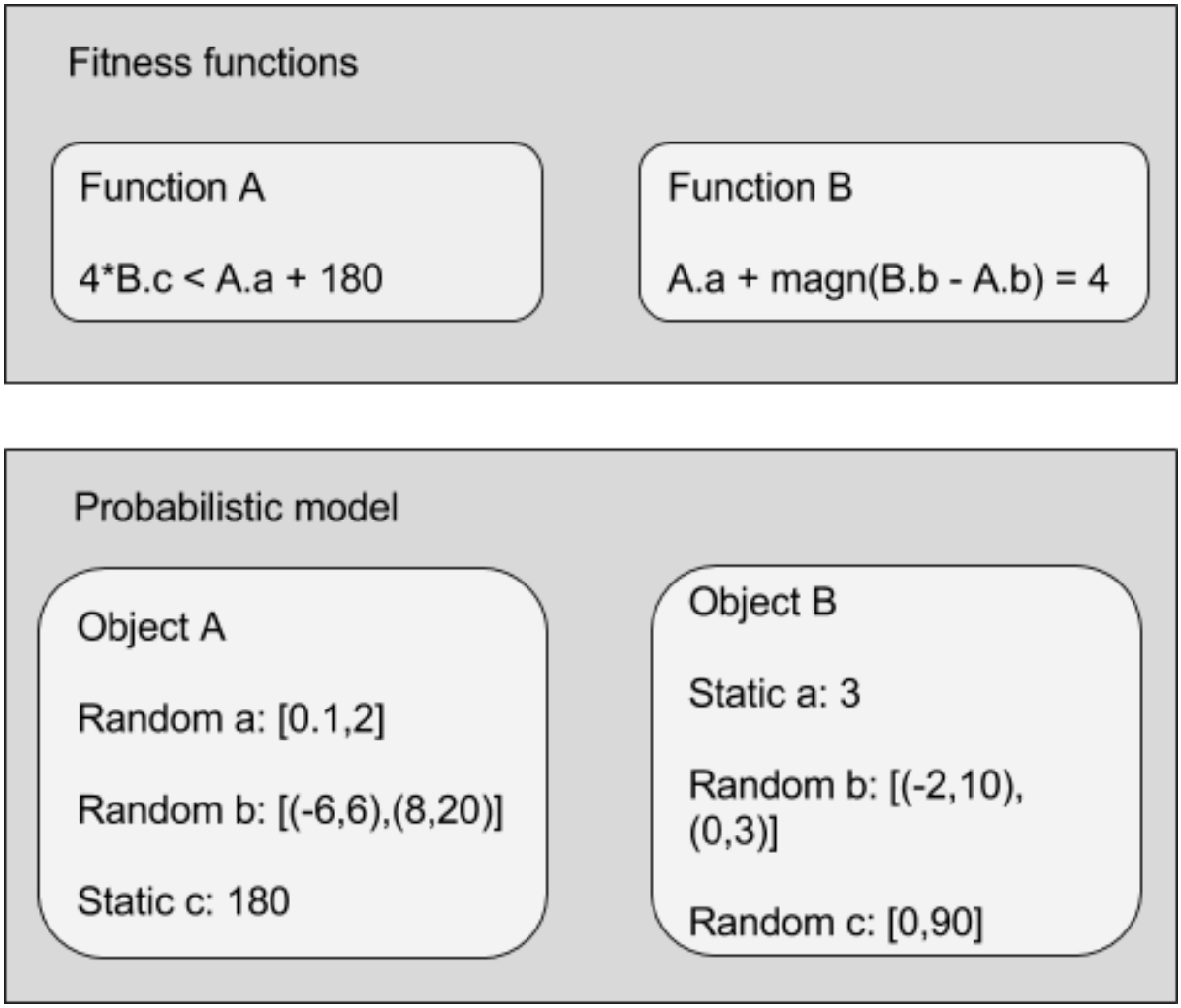


Figure 3.2: Bottom-up and top-down example

In terms of controllability, the automatic optimisation of the search space elevates the burden from the user to carefully design it. The user can define it in a coarse way, using the intervals, and let the the learning process handle it is refinement. This hyper-generator allows users to automate the construction (in this case search space declaration) of parts in their content generator over which they do not desire direct control. However, the user still has the possibility to directly define a variable' s distribution and exclude it from the learning process. Reducing the search space improves performance, increases convergence rate, by no longer sampling over values that always result in invalid samples [48].

3.2 Generative process

In figure 3.3 is given an overview of the generation process. The sections for the generative process of the hyper-generator correspond to the aforementioned components from search based approaches. The terminology has been slightly altered to make it more general and to set it apart from the terminology in search based approaches. The probabilistic model, mapping

and sampling respectively correspond to, genotype, genotype-to-phenotype mapping and search algorithm. The term fitness functions stays the same.

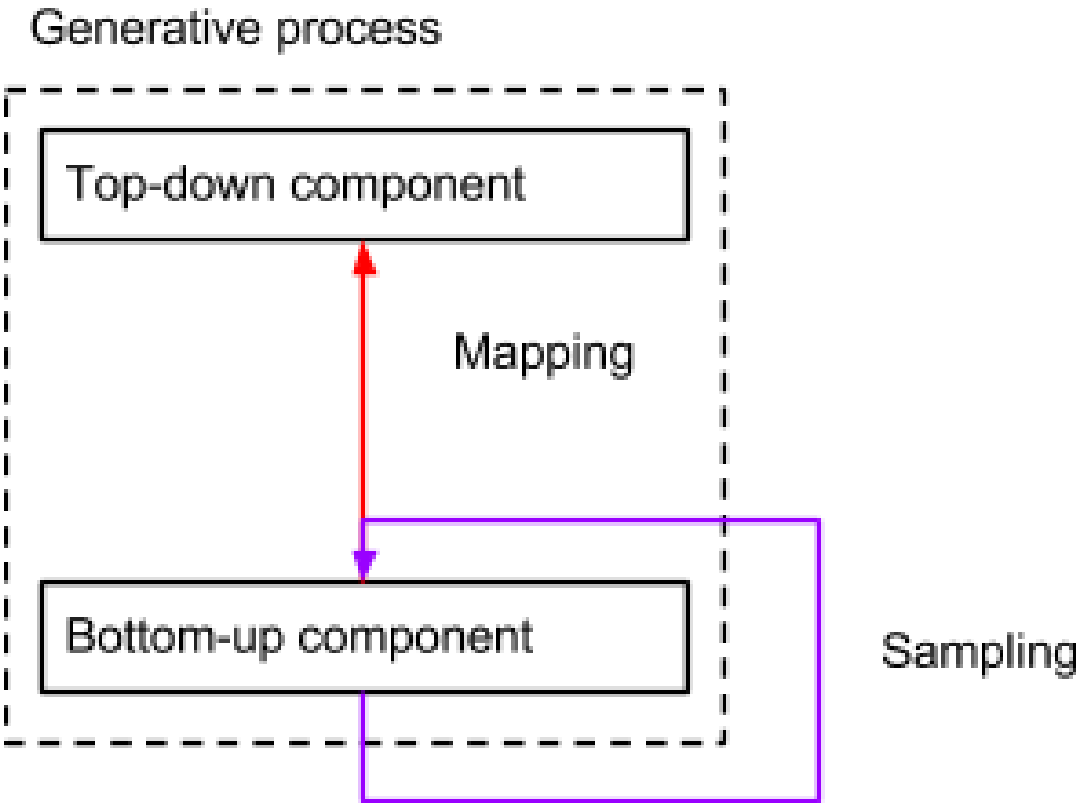


Figure 3.3: Overview generative process

3.2.1 Probabilistic model

I will elaborate on the probabilistic model, which defines the distribution of all the possible solutions "a priori", by translating its concepts to the use-case of scene-layouts. This does however not exclude the possibility for other layouts. In figure 3.4 the structure of the component is visualised. It is a hierarchical structure where each node represents an asset type. For each asset type it is possible to declare its variables. Additionally, child nodes in the hierarchy can declare their variables relative, by relating them to variables from the parent node with a function. This probabilistic model will be adapted in the learning process, upon which I will elaborate in the section 3.3.

Variable declaration

Variables are declared with a random distribution or static values, respectively called stochastic and deterministic variables. In the generation process each stochastic variable has a separate search space defined as a uniform distribution with real valued vector bounds. For example $position = (x, y)$ with $x \in [1, 4]$ and $y \in [4, 9]$. The fact that variables have real values is important because this implies a continuous search space, in contrast to the discrete search

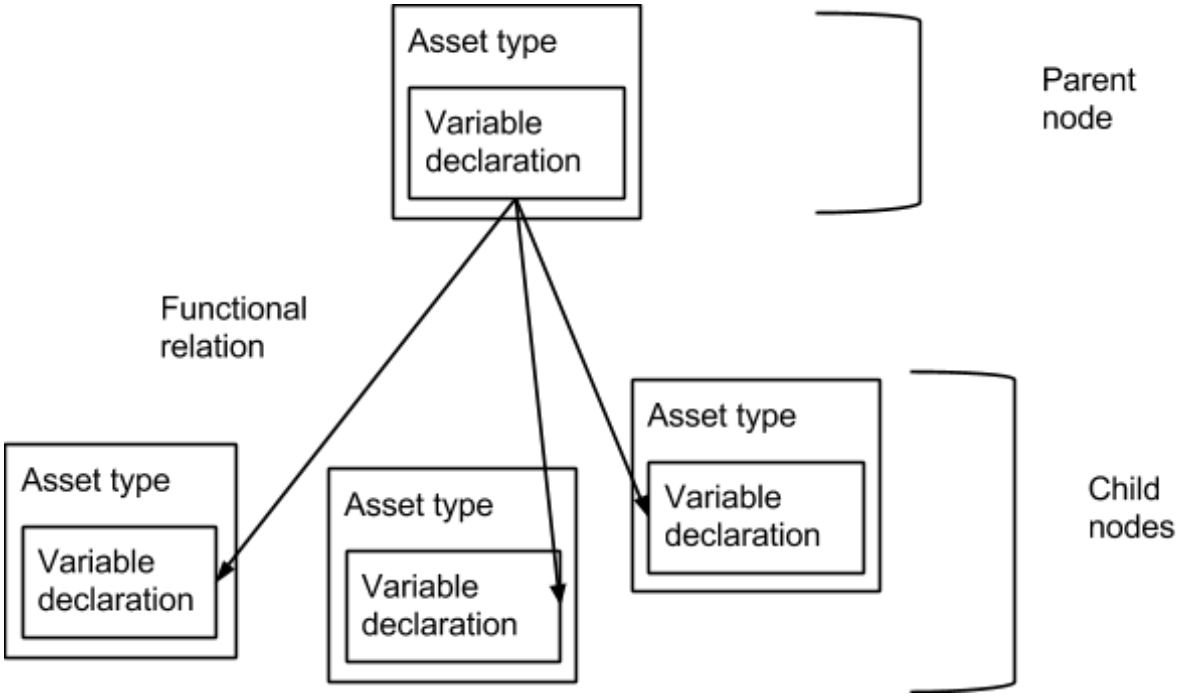


Figure 3.4: Bottom-up component

space of strings which are most often used in genetic algorithms. For which the limitations have been mentioned several times in the related work chapter. These bounds represent the outer limits of the uniform distribution from which the variable values will be sampled. In the first phase, all variables are declared by the user the search a predefined range. In the second phase, the learning process will automatically search for a more appropriate distribution (i.e. bias and range limits).

A variable with a uniform distribution has the maximum amount of uncertainty or least amount of information. But it also has the largest representative range, as there is no bias towards any particular value. As opposed to other more exotic distributions like Gaussians. Changing these distributions in the learning process will evidently affect the variables representative range. The new search space integrates the learned distributions which could have a smaller range than the user-defined uniform distribution.

Dependency tree

The variables are structured hierarchically as a dependency tree. A dependency tree is a directed acyclic graph where each node only has a single parent. Our system evaluates nodes in a single pass, because we want the probabilistic model to be purely constructive and therefore without iterations. This structure does not support circular dependencies, because with circular dependencies the nodes can no longer be evaluated in a single pass. There exists no valid evaluation order for the nodes. The user can also opt for a “flat” structure where all asset types and their corresponding variables are independent. This, however beats the purpose of the methodology, because I want to learn the relation between variables in child nodes and between parent and child nodes.

However, the variables themselves are not directly placed in the tree, but grouped in a node

which can have a semantic mapping to real world objects, in my case asset types. For example in figure 3.5, A chair with size between 50 and 70 cm, its position on an axis within an area of 2 by 2 meters, 5 meters from the left of a table. It is thus the chair (child) that will be placed according to the position of the table (parent). As my thesis focuses on generating open-

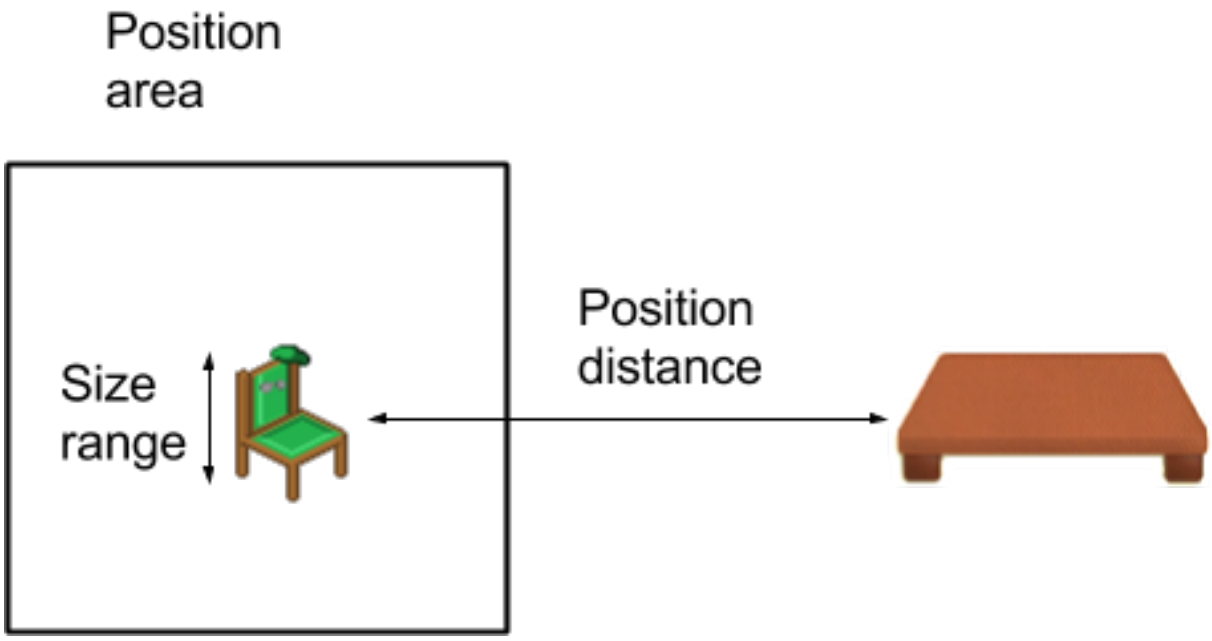


Figure 3.5: Dependency tree example

world layouts, the number of sampled asset types is a variable as well. This is declared in the mandatory variable "number of children" in each node. Which are stochastic or deterministic like the other variables, the values are however discrete.

The dependency tree is additional "a priori" knowledge which improves the generation process by optimising the search space using a "divide and conquer" approach. This reduces the dimensionality of the problem because the values of the variables in a node, and consequently the variable' s search-space, will only depend on the values of the variables in the parent node. And thus the variable search space is optimised by only taking into account its respective parent and sibling nodes. Sibling nodes are nodes of the same type with a common parent. Consequently this results in better scaling such that large layouts are solved in multiple stages.

As my thesis focuses on generating open-world layouts, the number of sampled asset types is a variable as well. This is declared in the mandatory variable "number of children" in each node. Which can be stochastic or deterministic like the other variables, the values are however discrete.

The dependency tree is additional "a priori" knowledge which improves the generation process by optimising the search space using a "divide and conquer" approach. This reduces the dimensionality of the problem because the values of the variables in a node, and consequently the variable' s search-space, will only depend on the values of the variables in the parent node. And thus the variable search space can be optimised by only taking into account its respective parent and sibling nodes. Sibling nodes are nodes of the same type with a common parent. Consequently this results in better scaling such that large layouts can be solved in multiple stages.

Functional relation

The function which relates parent and child variables can be used to declare a forward relation. For example in figure 3.6 when placing chairs, it makes sense to position them in a room according to the position of the table they belong to. The dependency is illustrated in figure 3.7. These semantics are completely up to the user. Chairs and tables can be spread over the whole

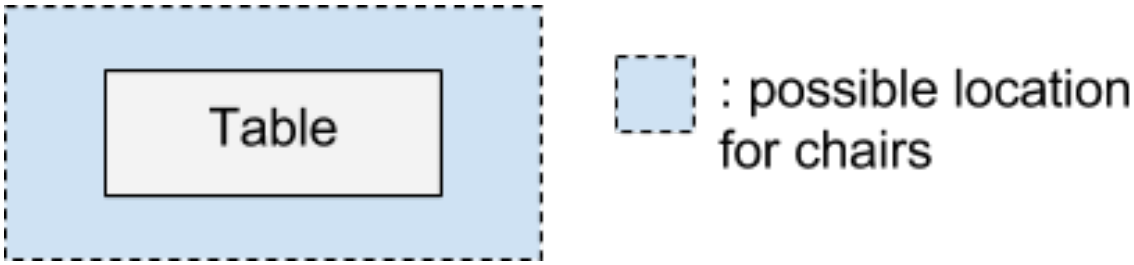


Figure 3.6: Example of possible location for chars around a table

room as well, without any hierarchy in the search space. The type of relations is restricted to affine transformations, including translation, rotation and scaling. This restriction is because it is possible to prove that affine transformations will not interfere with the learning process, which I will discuss further in the paragraph 3.3.2. When combining the variable densities with

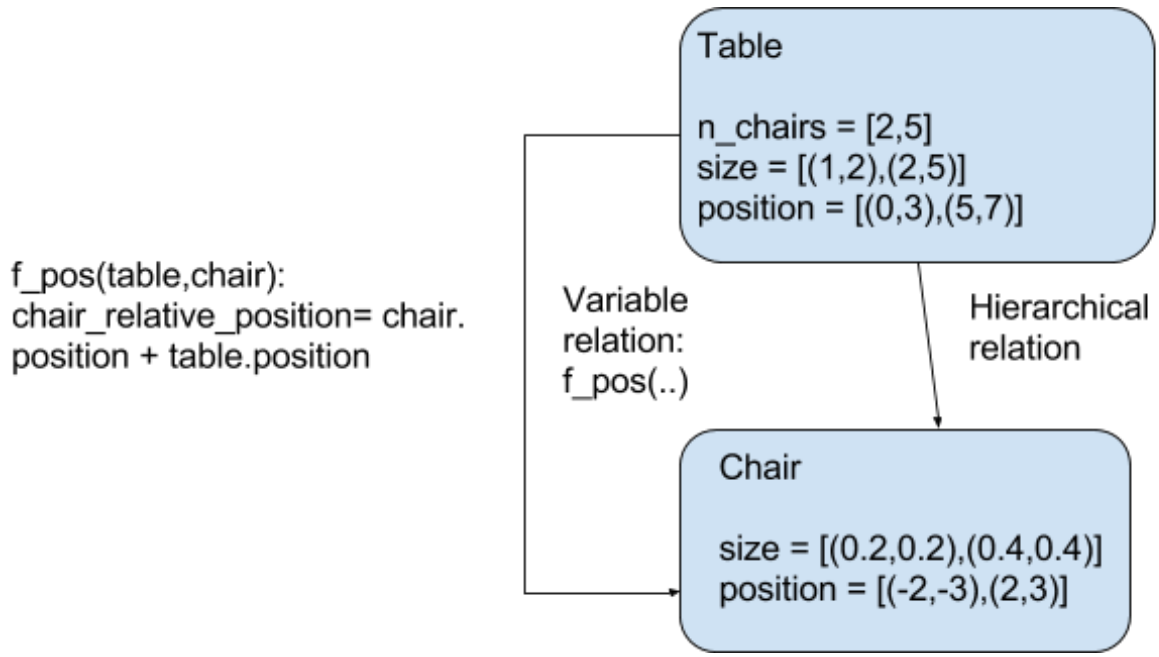


Figure 3.7: Example of relation between chair and table

aforementioned variable relations the generation process can be regarded as sampling from a

dynamic dependency tree. The sole difference with a regular dependency tree is that the number of nodes is not static but dynamic. This is due the fact that I am generating an open world with variable number of asset instances, the number of child nodes can be a random variables as well. In a dependency tree structure each parent is evaluated before its children. The evaluation of the tree, or a single iteration in the sampling process, requires that first the variable number of children is sampled. Once the the number of child nodes is known, that number of nodes is sampled relative to its parent.

When speaking of the sampling of a node, I mean the sampling of all its declared variables. Sampling will be elaborated upon in section 3.2.4. In [71] they formalize open world layout generation as sampling from a open world distribution. Static dependency structures like bayesian networks have been used as model for scene layouts in [72].

With respect to direct or indirect representation of the content, elaborated on in related work, the hierarchical model can be seen as a flexible way to allow both. Because the parent-child relations are user-defined, it is up to them to decide whether it is a semantic relation like the chair and tables or rather an additional level of abstraction over the child node variables, e.g. chair-group. If all variables of the child nodes can be deterministically derived from the parents variables values than the parent can be regarded as a more indirect representation of the children.

3.2.2 Mapping

In the hypergenerator the mapping is a simple 1-on-1 transformation of the values sampled from the probabilistic model into a representation for the fitness function to evaluate. For example position, rotation and size values into corresponding polygons. A complex and non-linear mapping, as mentioned in the related work, is necessary when the search space of the content has an indirect representation. However the focus of my thesis is on learning the influence of the fitness functions and not the mapping. In figure 3.8 is an example of a simple polygon mapping.

However, as mentioned in the introduction, these polygons are not the actual content which will be shown in a game. They are a simplified representation of the area covered by the content, for example a table, in the scene, for example a restaurant. The transformation from search space values into actual asset types, for example the 3D mesh of a table is another type of generation, called asset generation and will not be addressed in my thesis.

3.2.3 Fitness functions

The content is constrained in a top-down manner with a set of fitness functions. First I will discuss the different types of fitness functions with a few examples, afterwards I will elaborate on the requirements of the fitness function in a subsection.

Once a layout has been sampled the variable values are mapped to a certain representation. This representation can than be used to calculate the layouts “fitness”. The fitness function can be any injective function of the mapped representation to a real value. The fitness function informs the learning process which particular sampled layouts are desirable and which are not. However, there is no guarantee for the learning process to be able to optimise for a particular function. In the evaluation I will identify and test several fitness function used in the context of scene layouts.

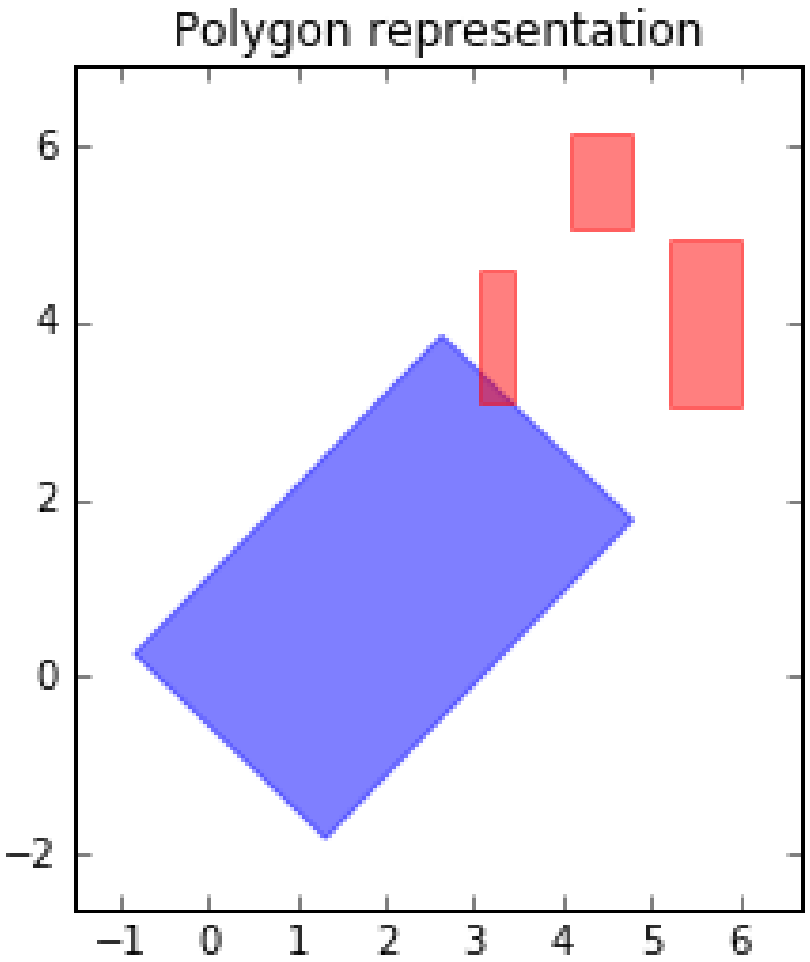


Figure 3.8: Example of polygon mapping

A fitness function implies a constraint between variables of the samples. For example by declaring that table and chair samples, represented by a polygon, should be aligned by their closest side, the function implies a constraint on the rotation and shape of the samples. Given the rotation and shape of the table it would be possible to derive the respective values for rotation and shape of chairs. Thus fitness function are distinguished based on the type of relation between instances they imply. In hierarchical terminology these relations are (1) pairwise between a parent and child node, (2) pairwise between siblings or (3) absolute over the complete layout. I will call those respectively parent-child, sibling and absolute fitness functions.

In the figures below are example constraints for furniture layouts originating from the work Merrel et al. These constraints are chosen in order to have a semantic mapping for them and will also be used in the evaluation, but our methodology is not limited to these constraints.

For example in figure 3.9. In order to facilitate traversal through a room (by player characters for example), depicted by the dashed line in the figure, the objects in the room should have a minimum distance from each other. This constraint is defined as pairwise fitness functions between siblings. Because each object only needs to measure its own distance to its siblings in order to meet the constraint.

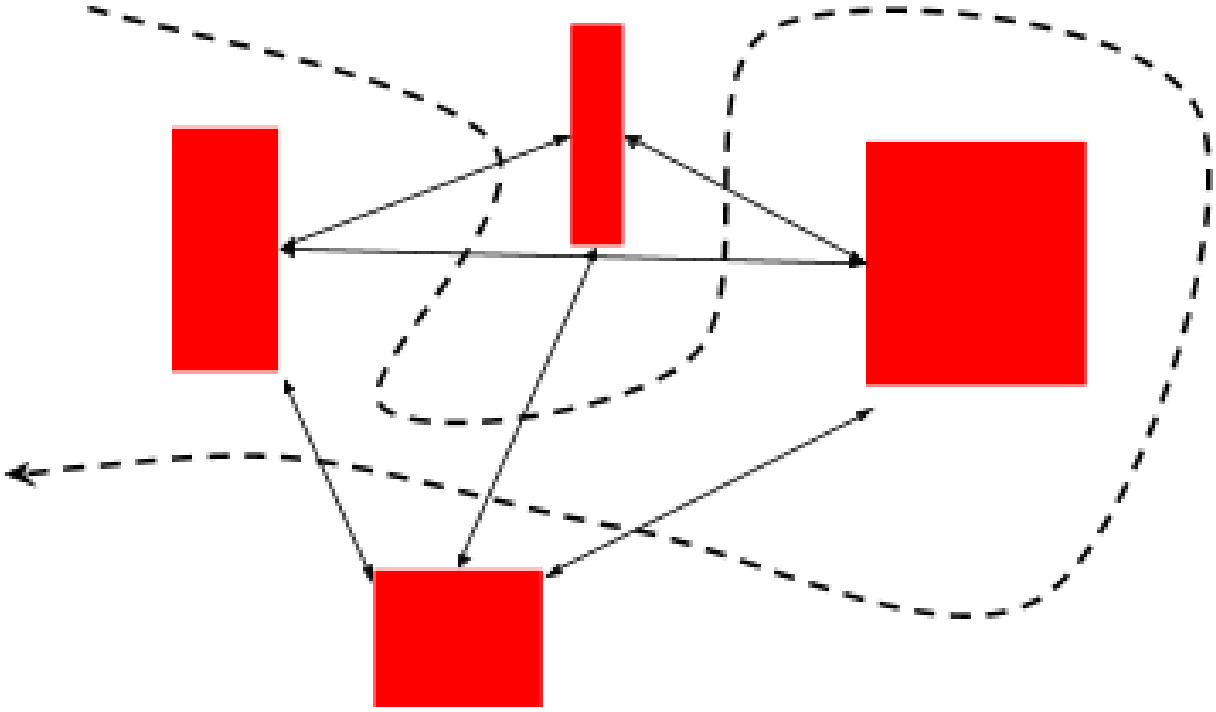


Figure 3.9: Minimum distance constraint

Next I will elaborate on how each type of fitness function is calculated. Firstly, the fitness between parent and child is calculated once for each parent-child pair. Secondly ,the sibling fitness is calculated once for each child, between the child and all its siblings. And finally, the absolute fitness is calculated once per sampled scene layout.

I distinguish the fitness function based on their relations to able to support separate fitness values for each child node. These separate fitness values as opposed to only one could influence the learning process, this will be elaborated upon in paragraph 3.3.2 .

Similar distinction has been made in the aforementioned paper by Yeh. et al. in order to construct the factor graph which encodes a scenes constraints, see in figure 3.10 below from Yeh. et al. The two shapes on the left represent tables with different number of plates. For these shapes they defined three constraints:(1) the occupied area factor, an absolute constraint, between both the table and the plates, (2) the inside factor, a parent child constraint and (3) the non-overlap factor, a constraint between siblings. With these constraints they build a separate factor graph for the case of 3 and 4 plates.

Requirements To be able to learn fitness functions in the learning process, there are some analytical requirements they have to meet. Firstly the function should be injective, which means that each input value has only a single corresponding function value. Because if a fitness function can have multiple values for a single given layout than the underlying model will treat the variation on the fitness function as noise. This is contrary to what I want to achieve, learning the distribution of layouts that have a single fitness function value, the maximum.

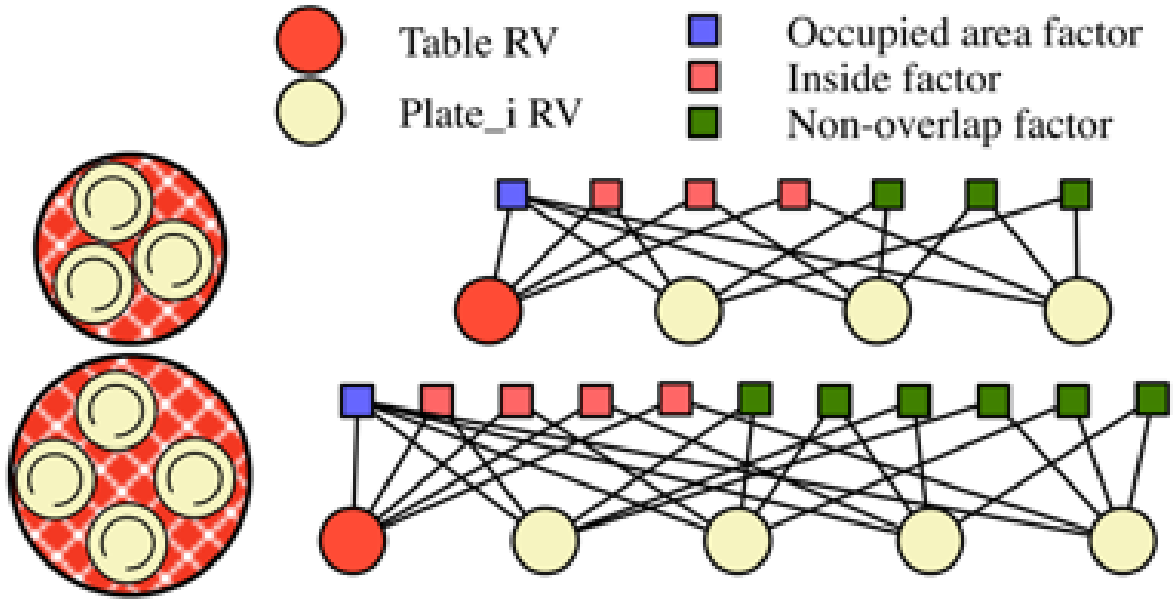


Figure 3.10: Example of three constraints on a table with plates

3.2.4 Pure random sampler

Once the search space of a layout has been defined, as a probabilistic model, it is possible to sample from it. In order to properly learn the search space distribution I need an unbiased subsample, representative for the actual search space. The most unbiased way of sampling would be uniformly. This is why evolutionary algorithms are less suitable for my methodology, because of the the unpredictable bias due to a certain crossover or mutation function [55].

However, even when sampling uniformly, an accidental bias can still arise if the amount of samples is too small compared to the search space variance. In the context of machine learning both previous issues are discussed under the term "bias-variance tradeoff". To avoid this bias when using the samples in the learning process, I will either sample a lot of times or use Poisson disk sampling (PDS). This method returns a random set of points with desirable properties; the points are tightly packed together, but no closer to each other than a specified minimum distance [**?**]. In the figure 3.11 can be seen that uniform sampling (right) can result in artificial grouping of points, a Jittered Grid (center) can offer a solution for this. The Poisson disk sampling (left) is however more evenly distributed over the sampled space. To sample the probabilistic model I need to sample from a dynamic dependency tree. Sampling a dynamic dependency tree is a recursive process and is visualised in figure 3.12. It starts with sampling the root parent node. Once the variables of a parent node are sampled, a parent node sample is created which contains its values (1). Secondly, the number of children (n in the figure) is sampled as well, if this number is a stochastic variable (2). If the child nodes variables are relative to the variables of the parent, the reference of the parent sample will be passed to the child nodes in order to calculate their respective variable values (3). Given the sampled number of children n, the n first child nodes will be sampled, given the parent samples (4).

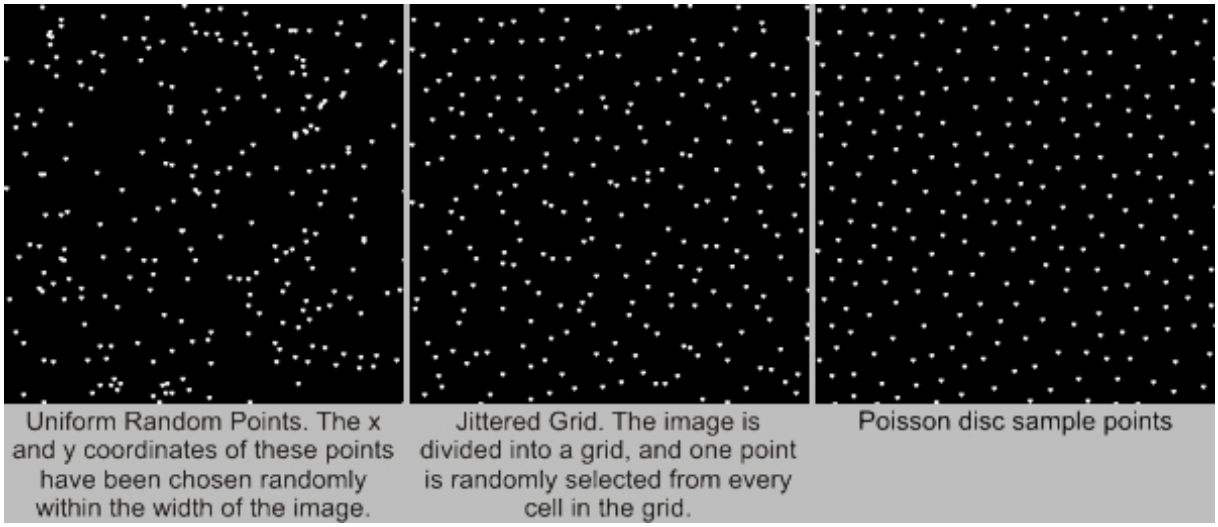


Figure 3.11: Comparison of different sampling approaches

3.3 Learning process

In order to optimise the search space defined in the generation process I will generate samples of each asset type using the generation process, calculate each sample' s fitness and use this fitness as guide to optimise the search space.

This section first formulates the root learning problem of my approach to be able to learn from the generation process. Secondly I discuss the components of the learning process. The probabilistic model, also found in the generative process, but now in the context of learning. The learning algorithm which trains the probabilistic model. The pre-processing which extracts the data required for the learning algorithm from the samples generated in the generative process. And the retraining component which decides whether previous steps need to be repeated. The third section elaborates on an adaptation of the sampling discussed in the generation process. The sampling after the learning process is different from the sampling in the generation process because the probabilistic model will have a slightly different structure.

3.3.1 Learning problem

The probabilistic model for learning needs to be compatible with the probabilistic model of generation because I want to offer a "white box" solution. Where, after the learning process, the user still has the ability to make changes to the generator after it has been optimised in the learning process. This in in contrast with Alexey et al. [88], where they use a convolutional neural network as probabilistic model to generate 3 dimensional models for chairs. However, it is practically impossible for the user to directly edit the generator, the neural network, because this would require understanding the behaviour of a neural network with up to 10 layers each containing up to a 1000 neurons.

In order to share the probabilistic model between the generative and learning process the training of the probabilistic model needs to be decomposed. First I elaborate on the decomposition of the hierarchical structure in the probabilistic mode. Therefore the learning process is solved with standard learning methods. Secondly, I discuss how the learning method is used on the samples from the generation process.

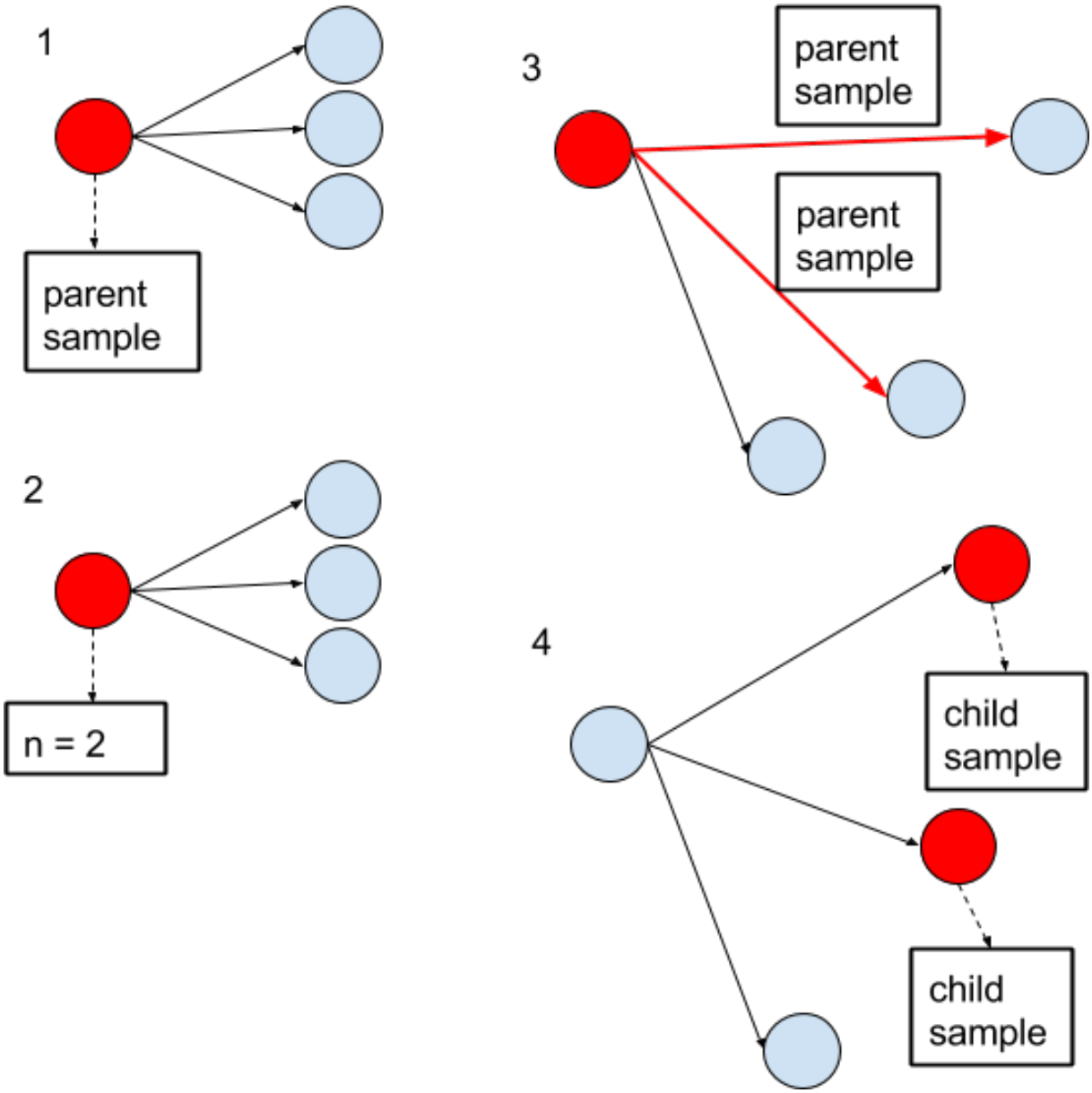


Figure 3.12: An example of the first steps in the recursive sampling

Additionally, the learning process is limited to learning the relations between variables, given a user-defined hierarchy of these variables. This is in in contrast to learning the hierarchical structure between random variables itself. Learning the structure of a hierarchical model has been done in the context of building layout generation by Merrel et al. [79]. However, I focus on solely learning the relations between variables and will add structure learning in a later phase, if time permits it.

In the next subsection I will firstly discuss how the generation process is decomposed. Secondly, I elaborate on the learning method itself.

Hierarchical relations as conditional probabilities

I want to learn two types of relations. The first is between the variables of a parent and its child nodes, the second between the variables of sibling nodes, child nodes with a common parent. In order to learn these relations I will formulate the probabilistic model as a probabilistic *graphical* model. Which is a probabilistic model with a graph structure to represent the conditional dependencies between random variables. Therefore, the aforementioned relations will be modelled as conditional probabilities, called conditional relations. Combining both conditional relations allows me to answer the query "Where to place the second chair, given the table and the first three chairs already placed".

A conditional probability is a measure of the probability of a random variable given that another random variables has a certain value. If the variable of interest is A and the variable B is known "the conditional probability of A given B" is usually written as $P(A|B)$. It is defined as the quotient of the probability of the joint probability of variables A and B, and the probability of B:

$$P(A|B) = \frac{P(A \cup B)}{P(B)}$$

The conditional probability will model the parent-child relation as the probability for the variables of a child given the variables of the parent. And the sibling relation as the probability for the variables of a child given the variables of the siblings preceding it in the sampling process. This fits in the generation process by first generating the parent sample and using this sample to condition the densities of the child variables. Additionally, the children will be sampled in a fixed order. To condition each density of the child on the available variables from its preceding siblings in the generation process. I will estimate the conditional relations, illustrated in figure 3.13, between the parent (circle) and the siblings (rounded squares). Readers familiar with

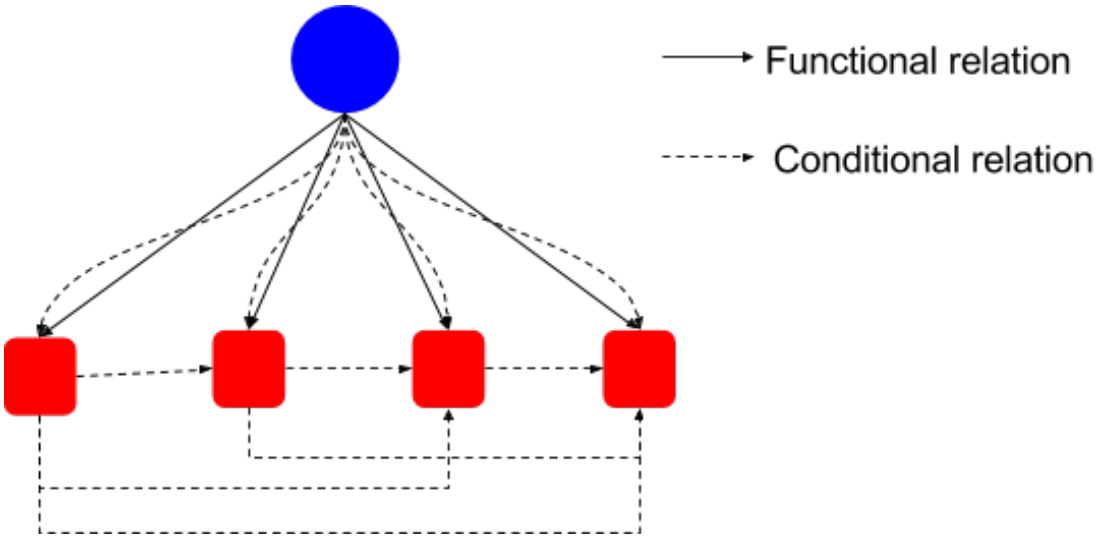


Figure 3.13: Example of all possible relations between parent and child nodes

probabilistic graphical models will have noticed similarities between the probabilistic model in figure 3.13 and a Bayesian network. A Bayesian network is a probabilistic graphical model which represents the conditional dependencies using a directed acyclic graph. There are however 2 key differences. First, the additional functional relation in my probabilistic model. And secondly, the restriction on the edges. Whereas in a Bayesian network the edges can connect any two nodes, as long as no cycle is formed, my model restricts edges to a single parent node, its children and between siblings, children which share the same parent node.

Not all variables of a child need to be included in the training process. For example, it is allowed to only train the density of the position of the chairs and not their rotation. I will call the child variables for which the probability is estimated, the variables of interest (VOI). The VOI do not have to be the same as the variables of the parent. For example, I can estimate the probability of a chair's position given the rotation of the parent table. The conditional probability of the third child is given as example below.

$$VOI = childvariablesPV = parentvariablesP(VOI_{child2}|PB, VOI_{child0}, VOI_{child1}) \quad (3.1)$$

However, in the generation process all children are generated at once. But it is intuitively clear that this is no longer possible. By adding the conditional relations between the variables of siblings, all children need to be generated in an ordered sequence. Which, as aforementioned, allows me to explicitly query for the variables of child with index n, given the variables of the $n-1$ previously sampled siblings.

Another issue which needs to be addressed in order to integrate the learning process with the generative process. Is the issue of a dynamic number of children in the hierarchy of the probabilistic model. The probability of a random variable is mathematically modelled by a probability density function, also called a density. This function has a fixed dimensionality and can therefore only model a fixed amount of random variables. When the number of children changes, so does the number of variables I want to estimate the probability of. The problem can be solved using multiple instance learning (MIL) [89]. In MIL, training samples are grouped in bags containing many instances. The term originates from text analysis where the frequency of each word is used instead of the text itself. This reduces the MIL problem into a single instance problem, because there are is a fixed amount of unique words for any text or set of texts. The MIL technique I will use is based on work by Li et al.[90] where they want to track an unknown number of objects in video frames. They propose a method to separate moving objects from the background by using multiple Gaussians to describe the objects. This method has shown to be stable and efficient in dealing with rotation, pose and appearance changes. I will separate samples from the model into independent groups, each only containing samples with a certain amount of child nodes. This also means that the estimation of the probability will have to be done separately for each group, which results in a separate probability density function for each group. However, these separate densities still need to be combined such that the generation process can sample from a single probabilistic model. I will address this issue in the paragraph 3.3.2 In Yeh et al. they address a similar issue for generating what they call "open world" layouts with constraints. Open worlds in contrast to closed world can have a variable amount of objects. The crucial difference with my methodology is their way of encoding constraints. They encode their constraints with factor graphs, I refer to the related work for an elaborate explanation. These factor graphs are an undirected graphical model, which is not usable for my hyper-generator because as mentioned in subsection 3.2.1 the generation process requires a directed model. In a directed relation the order of the variables in the relation is fixed "X comes before Y". Given a fitness value I will approximate this forward relation between a previous sample' s variable and the next. This learned forward relation, in the form of a conditional probability, should however not be confused with the user-defined functional relation in the generation process 3.2.1. The functional relation is a deterministic function, given a value the function return its transformed value. Whereas the learned conditional density is a probabilistic function, given a value it returns a range of possible values in the form of distribution from which values are sampled. The deterministic functions should be leveraged by the user to define the search space bounds from which all desired child samples, given a parent samples, are generated. The learned directed relation will further refine this space by looking for the search space' s form within these bounds which contains as much children with good properties and as few without. However, the user should be aware that all learning comes with a computational price. The fitness functions can be expensive to compute, hard to enforce due to conflict with other fitness and there' s no guarantee an optimised distribution will be found.

Supervised density estimation

Not only do I need to learn the conditional density of the variables, which reflects their hierarchical relation. But I also need to learn these densities in such a way that a variable's density should have a higher probability for values which result in higher fitness. The learning method I will use for this is called supervised density estimation. Supervised learning, in contrast to unsupervised learning, does not solely try to learn the structure of the data but requires additional labels for its data. In my methodology the label informs the learning method how important a certain data point is, given its fitness, for the density estimation. Supervised density estimation has been used extensively for the purpose of optimizing a heuristic search space [91] [86] [92]. I will elaborate on how the label for a sample, called fitness label, is derived from its fitness is in paragraph 3.3.2. I will explored two techniques for supervised density estimation, weighted density estimation and probabilistic regression, elaborated upon in the subsection 3.3.2.

3.3.2 Components

In this section I will elaborate on each component of the learning process. Firstly, the probabilistic model which is a common component between the learning and generative process. Secondly, the learning algorithms which are used to train the densities in the probabilistic model. Thirdly, the pre-processing required to apply the learning algorithm on the samples from the generative process. And finally the retraining component which decides whether previous steps need to be repeated.

The probabilistic model

Similarly is in the generation process the probabilistic model is defined by densities and a hierarchical structure relating them. However, in the generation process the densities are initially uniform whereas in the learning process the densities are a statistical model with parameters which need to be estimated. I first discuss this parametric model and secondly the hierarchical structure.

Gaussian mixture model In general one can choose between parametric and non-parametric models, for example gaussian kernel density functions. The kernel methods are a non-parametric way of density estimation and are essentially a smoothing problem based on a subset of the population, the method needs to stores this subset. However, for integration in the generation process I need to compute the conditional density at every sample step. And this computation is more expensive because in contrast to parametric model it scales linearly with the amount of data used [93]. Additionally I need to calculate the marginal distribution, discussed in paragraph 3.3.2, which has the same computational complexity. Therefore I chose for a parametric model, to be more precise a Gaussian mixture model. Gaussian mixture models (GMM) represents a parametric probability distribution as a weighted sum of multivariate Gaussians. They are commonly used as a parametric model for the probability density of continuous measurements. A gaussian mixture is capable of approximating any continuous n-dimensional function and provide a relatively lower computational cost of conditional and marginal distributions than the kernel density functions. A GMM has three parameters. (1) A set of weights WGMM and a set of mean vectors μ for each Gaussian in the mixture and a covariance matrix Σ which defines the covariance between each Gaussian. In figure X is an example of a GMM. In the next subsections I will first discuss the hyper-parameters which can influence the estimation of the model parameters. Secondly, I discuss properties of the GMM which are part of the motivation for choosing the GMM as statistical model.

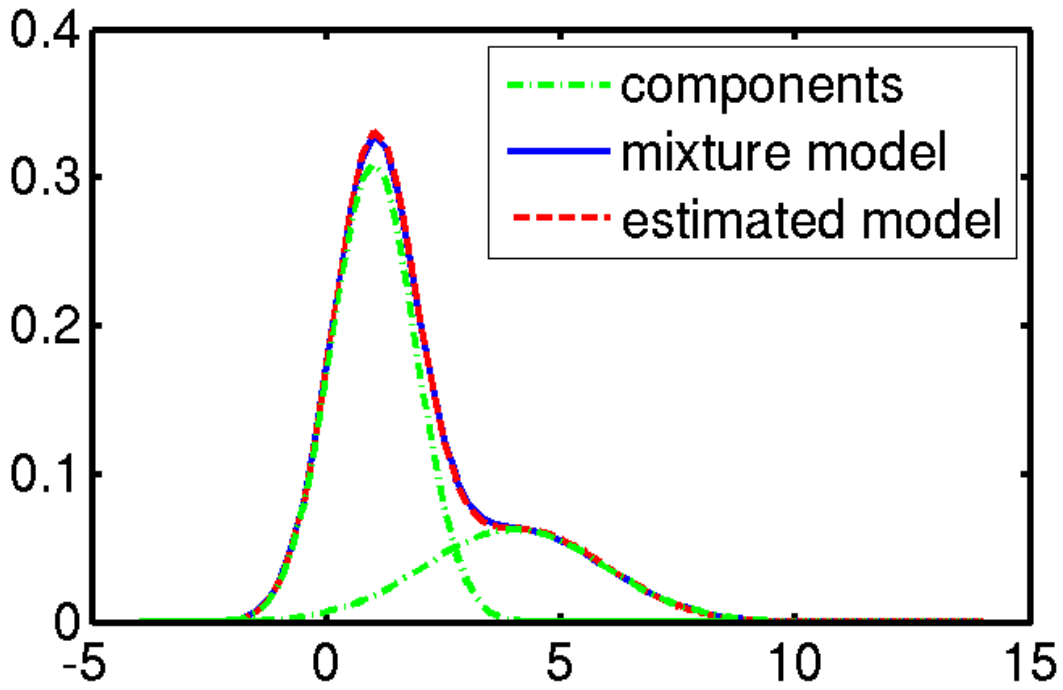


Figure 3.14: Example of a Gaussian Mixture model

GMM parameter estimation Before estimating the parameters of a GMM, I have to define how much components, Gaussians, will be in the mixture model and what the minimum covariance during parameter estimation will be. The number of components is an important parameter as it is an indication of complexity of the model and thus over-fitting. The minimum covariance is the floor on the diagonal of the covariance matrix, which is used to prevent over-fitting. This means that if during the parameter estimation the covariance of a single components drops below the minimum, the estimation process is stopped and the parameters from a previous iteration are used.

GMM Properties The marginal distribution of a subset of variables in the probability of a larger set of variables gives the probability the variables in this subset without reference to the other variables. The variables in this subset are called the marginal variables. The purpose of marginalisation will be elaborated upon in 3.3.2. However, it is important to mention that marginalising is a computationally cheap operation. In order to marginalise certain dimensions in a Gaussian mixture one marginalises these particular dimension for each Gaussian in the mixture. The marginal distribution of a multivariate Gaussian is simply calculated by dropping the irrelevant dimension, the ones that need to be marginalised out, from the mean vector and covariance matrix. As mentioned in section 3.3.1 I need to calculate the conditional probability of random variables. To condition a GMM you need to calculate the conditional distribution for each Gaussian component in the mixture and recalculate the weight of each component based on the values of the conditioned variables. In listing X I show the steps for conditioning a multivariate Gaussian. In none of these step was it required to retrain the GMM, its computational complexity only depends on the dimension of the Gaussian and is independent from the size of the training data.

If N-dimensional x is partitioned as follows

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} \text{ with sizes } \begin{bmatrix} q \times 1 \\ (N-q) \times 1 \end{bmatrix}$$

and accordingly μ and Σ

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix} \text{ with sizes } \begin{bmatrix} q \times 1 \\ (N-q) \times 1 \end{bmatrix}$$

$$\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix} \text{ with sizes } \begin{bmatrix} q \times q & q \times (N-q) \\ (N-q) \times q & (N-q) \times (N-q) \end{bmatrix}$$

then, the distribution of $x1$ conditional on $x2 = a$ is multivariate normal $(x_1|x_2 = a)$ $N(\mu, \Sigma)$ where

$$\bar{\boldsymbol{\mu}} = \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\mathbf{a} - \boldsymbol{\mu}_2)$$

and covariance matrix

$$\overline{\boldsymbol{\Sigma}} = \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}.$$

[94]

In order to support the functional relation in the generative process 3.2.1, the parameter estimation of the GMM needs to be invariant to these functional relations. Which is why I restricted the functional relations to affine transformations. The affine transformation of a GMM is the affine transformation of its components. An affine transformation is defined as $y = c + Bx$ and includes translations, rotations and scaling. If the x vector is normally distributed with mean vector and covariance matrix . Than y is normally distributed with mean vector μ and covariance matrix Σ, i.e. $(x \sim N(\mu, \Sigma))$. This implies that a GMM trained on features $(x_1, x_2, x_3, ...)$ can be transformed with an affine transformation to be a valid GMM for the affinely transformed features. For a more formal and elaborate proof of all Gaussian properties I refer the reader to Schon et al. [95].

Hierarchical structure As mentioned in section 3.3.1 a single probability density function is not enough to learn the densities of variables in the generative process. I need a single GMM for each possible number of children in the layout. The number of GMMs required in this approach would scale exponentially with increasing levels of depth in the hierarchical structure of the model, each parent can have a varying amount of children which in turn can have a varying amount of children as well. For this reason I will model the hierarchy level per level, but each level will be related through the conditional relations between parent and child nodes, elaborated upon in section 3.3.1 .

In this section I will discuss possible approaches to reduce the amount of GMMs which need to be trained to model a single level in the hierarchy. The first approach is to condition the child variables only on a limited amount of siblings. The second, will leverage marginalisation to avoid the need to retrain a different GMM for each child.

Variable sibling order Instead of learning a different model for each number of children. I propose an optimisation which relies on the independence assumption between the variables of the child and its siblings past a certain order. This principle defines the process of sampling children as sampling from a variable order Markov chain. In a Markov chain each state is assumed to only depend on the previous states in the chain. I will assume that each child is sampled by conditioning on only a certain number of previously sampled sibling nodes and not all of them. This order is called the sibling order. By using the same GMM for multiple child nodes, child nodes with the same sibling order, in the hierarchy I the number of required GMMs that need to be trained.

In the hyper-generator it is possible to define the sibling order for each number of children separately, as a sequence of orders. For example, the sibling order sequence [0,1,2,2,2,3] visualised in figure 3.15, implies that the first 3 children are conditioned on all previous siblings. However, the 3rd and 4th child only depend on the 2 last sampled siblings and the last child depends on the 3 previous siblings. The black dotted line represent the conditional relation between child and parent nodes. It is intuitively clear that the sibling order is never larger than the available amount of previous sibling samples. I will evaluate the influence of different sibling order sequences on generated content.

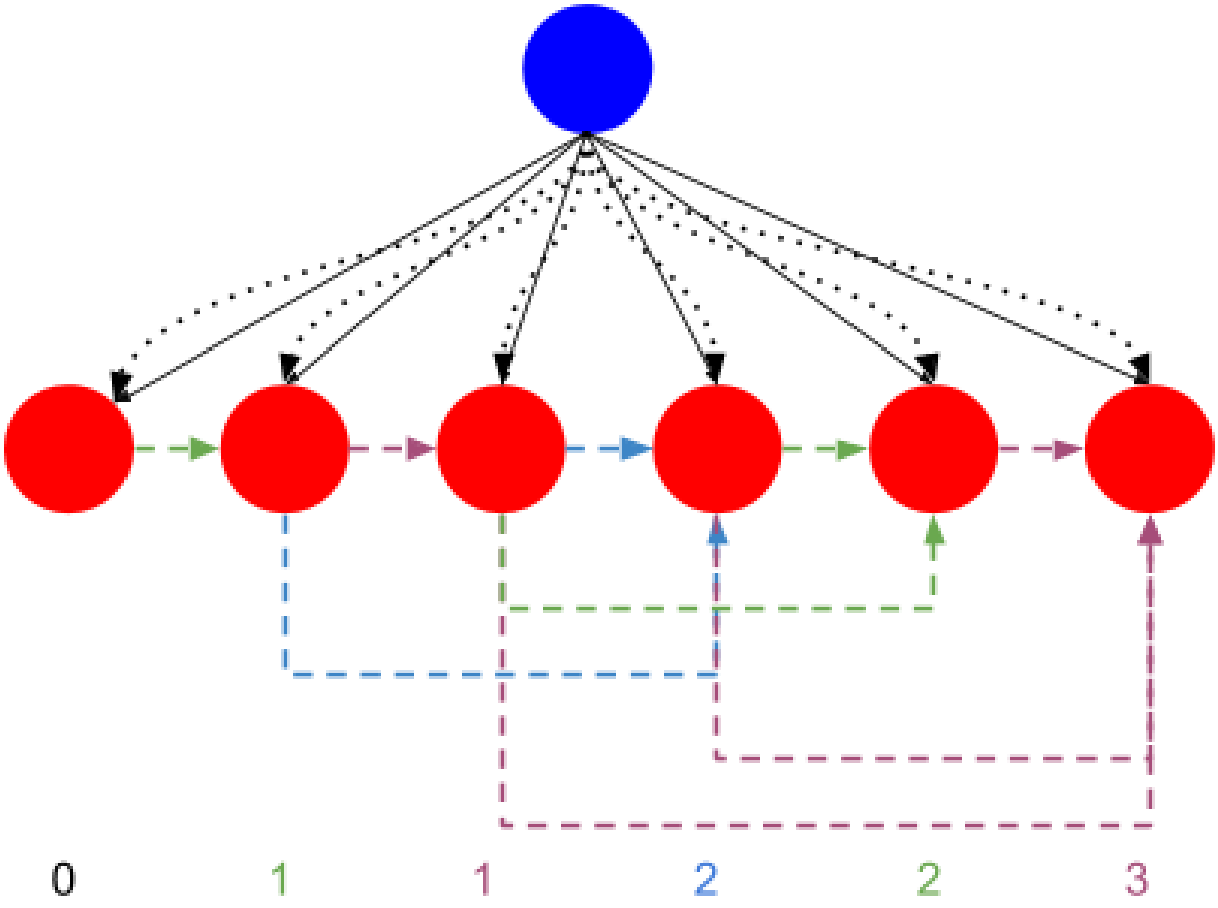


Figure 3.15: Example of a hierarchical structure given variables sibling order sequence [0,1,1,2,2,3]

Marginalisation I want to further reduce the amount of GMMs. Therefore, I make the assumption that the density which models the variables of a child conditioned on its n previous siblings can be marginalised from the density of a child which is conditioned on a higher number of siblings. For example, I assume that in a situation where someone wants to place a chair around a table with already 3 chairs around it. The optimal place for the 3rd chair similar to that of the 4th chair. Concretely, this would mean that only the density for the 4th chair needs to estimated and the the density for the 3rd chair is approximated from it. In general, given the GMM that models the joint distribution for child with order n and sibling order so, from which I calculate the conditional distribution of its variables. This joint distribution GMM_n is conditioned on the variable values of the previous $n - so_n$ sibling samples. I assume that the joint distributions for each child with sibling order $m < so_n$, can be approximated by simply marginalising the first m siblings, their respective variables, from the so_n order distribution. I will evaluate this assumption by comparing different degrees of marginalisation. An example of one marginalisation from a GMM for sibling order 3 to sibling order 2 is given in figure 3.16. However, if this would, for example, be applied to the joint distribution of a 5 child scene. In

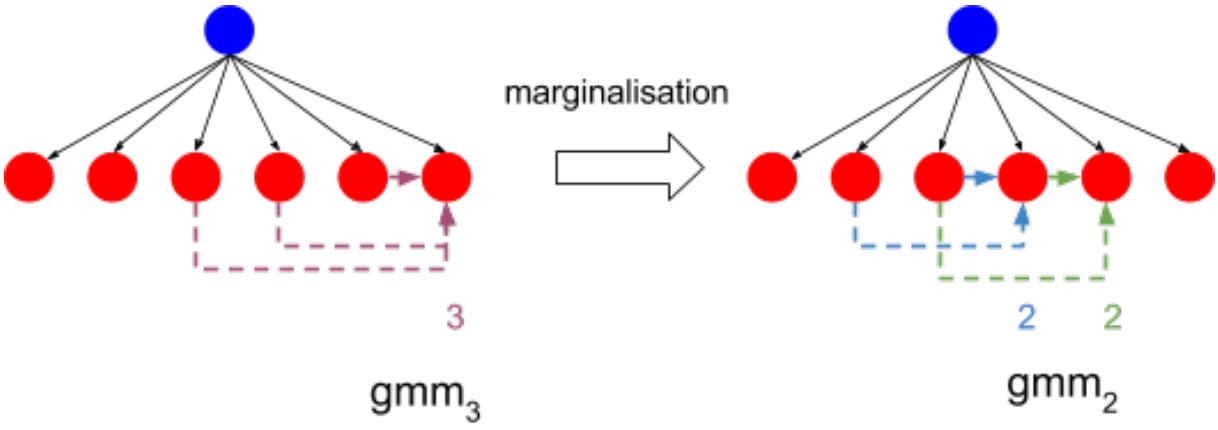


Figure 3.16: Example of hierarchical structure with marginalisation

order to calculate the joint distribution of the first child. This would imply that the distribution of the first child will depend on the fitness of the 5 child scene. Because in the supervised density estimation of n children, the fitness is calculated for the case of n child objects in the scene. At first this might seem a desired property, but the variance of the distribution will be too small This is because most often the fitness functions are harder to optimize with an increasing number of objects in the scene.

For example, compare the placement of an additional chair (while avoiding collision with other chairs) in an empty room with its placement in a room full of chairs. The marginalised distribution will approximate the distribution of the first chairs for which the next N are also well placed. The surface which will be considered to place the first chair (its positions distribution) will be much smaller than if I would simply place a chair without having to take previously placed chairs into account.

The variance of a distribution describes how far samples are spread out relative to the mean of the distribution and thus relates to the expressive range of the generator. I want to offer the user a flexible trade-off between the computational complexity of training multiple GMMs and the reduction on representative range it could imply. Therefore, I will allow the user to define, for each order ranging from 0 to the maximum sibling order, whether it should be trained separately or derived from a higher sibling order distribution. The exact influence of the sibling order list on the generated content, including its expressive range, will be evaluated

Learning algorithms

To estimate the parameters of a statistical model, a GMM in my case, I will use the most widely used parameter estimating method in machine learning called maximum likelihood estimation (MLE) [96]. I refer to paragraph 3.3.2 for the parameters of the GMM. This technique has proven to give reasonable results, it is however out of scope of this thesis to evaluate different parameter estimation techniques. The technique of maximum likelihood will fit the model parameter values such that it maximizes the likelihood function given the data and our model. By intuition the likelihood function maximizes the "agreement" of the model with the data, the data is in my case the variables of the layout. Such that the data sampled from the model with these parameters will closely resemble the originally observed data.

There are various algorithms for finding the MLE, I use the most common method called Expectation-maximization (EM). My motivation is the same as for choosing the MLE for estimating the model parameters, it is outside the scope of my methodology to evaluate other approaches.

I will elaborate on EM and how it is adapted to use it for supervised density estimation in the following subsection. The second technique I will discuss is called probabilistic regression, which uses standard EM, but differs from the first technique by estimating the labels jointly with the data. In the third subsection I elaborate on the parameters of EM.

Weighted density estimation To be able to incorporate the fitness labels into the model parameter estimation, I will use a technique which adds an extra layer over the likelihood function of MLE, called weighted density estimation (WDE). This has been practically used before, for example in a probabilistic modelling library called Pomegranate [97]. Conceptually, this means that not only the structure of the data will define the model parameters but the fitness label as well. Such that a variable's density should have a higher probability for samples which result in higher fitness.

Whenever the likelihood is calculated for all the variables in the layout, the result is multiplied with the fitness label. This fitness label combines all fitness values of a single layout in a single value by multiplying all normalised fitness values of this layout. Similar likelihood for a scene layout has been used by Yeh. et. al in their monte carlo method.

EM is an iterative method which performs two steps in each iteration. The first is the expectation (E) step, which calculates the expected value, for each sample in the data, of the log-likelihood given a current estimate of the parameters. The initial estimate, before iterating, of the parameters are random or tailored to the problem at hand. In the E-step the log-likelihood is used because it reduces the potential for underflow, due to very small likelihoods. Additionally, logs permit converting a product of factors into a summation of factors, which reduces computational cost in some cases. The next step, called the maximization (M) step calculates the parameters which maximizes the estimated log-likelihood function calculated in the E-step. Consecutively the found parameter estimates are used in the next E-step. The weights are integrated in the E step, after calculating the estimated log-likelihood for each data sample these are multiplied with their respective weight.

Given a set X of observed data, their corresponding weights W_X, a set of missing values Z, and a vector of unknown parameters , and a likelihood function $L(\Theta; X, Z) = p(X, Z|\Theta)$. The EM algorithm searches for the MLE of the marginal likelihood by iteratively applying the following steps:

E-step:

$$Q(\Theta|\Theta^{(t)}) = E_{Z|X,\Theta^{(t)}}[logL(\Theta, X, W_X, Z)] \tag{3.2}$$

M-step:

$$\Theta^{(t+1)} = argmax Q(\Theta|\Theta^{(t)}) \tag{3.3}$$

Probabilistic regression The other technique I experimented with is probabilistic regression. In machine learning regression is an approach for modeling the relationship between a dependent variable y and one or more independent variables denoted X. In probabilistic regression this relation is modelled as the conditional distribution of Y given X ($P(X|Y)$) [98]. I will use probabilistic regression to estimate the density of layout variables V given their fitness F ($P(V|F)$). Intuitively it seems appropriate to always condition on the largest possible fitness value. I will evaluate if changing the fitness values which are used to condition on, called fitness targets, influence the content of the hyper-generator.

This technique was partially motivated to address the problem of multiple fitness functions. Because in contrast to WDE I do not have to reduce the fitness of a layout to single value. I can train on the full fitness value vector, which contains a separate fitness value for each pair of child and fitness function in the layout. The multi-dimensionality of the fitness functions will be elaborated upon in section 3.3.2.

Parameters of the learning algorithm In the learning process EM parameters are fixed to a default value after initial experimentation and were not varied during evaluation. I have found their influence to be minimal and therefore decided that the complete evaluation of these parameters is out of scope and belongs in literature dedicated on parameter tuning of these algorithms. The first parameters is a convergence threshold. EM iterations will stop when average gain in log-likelihood is below this threshold its default value is $1e-3$. The second is the number of EM iterations to perform (default 100). And the last is the number of initializations (default 5), from which the best results is kept

Pre-processing

In order to use the learning algorithms, the samples from the generation process need to be transformed in proper training data with a fixed format. First, will give an overview of the of the complete methodology, with focus on the data flow of the pre-processing. Secondly, the process of generating the training data. Thirdly, I discuss the different possibilities of transforming this data, which will be evaluated, in order to positively influence the results of training the GMM.

Overview In figure 3.17 an overview is given of the pre-processing. It visualise the data flow of the training data, containing fitness data (dashed arrow) and asset data (solid arrow), from the generation process into the preprocessing stage and to finally pass it to one of the training algorithms of the probabilistic model. The dashed/solid arrow visualises the joined fitness and sample data. The dotted line visualises other data-flow relevant for the generation process but not this section.

Generating training data In general when trying to estimate the density of the data, the data is a matrix of the form $n_{samples} \times n_{features}$, where $n_{features}$ is the dimension of data you want to learn and $n_{samples}$ the number of generated samples in the data. In machine learning a feature can be any measurable variable of the observed objects or any derivation from these variables. Given the sibling order n, the features distribution in my case will be the joint distribution of variables from the parent and $n+1$ children. I will elaborate on the data format in the next subsection.

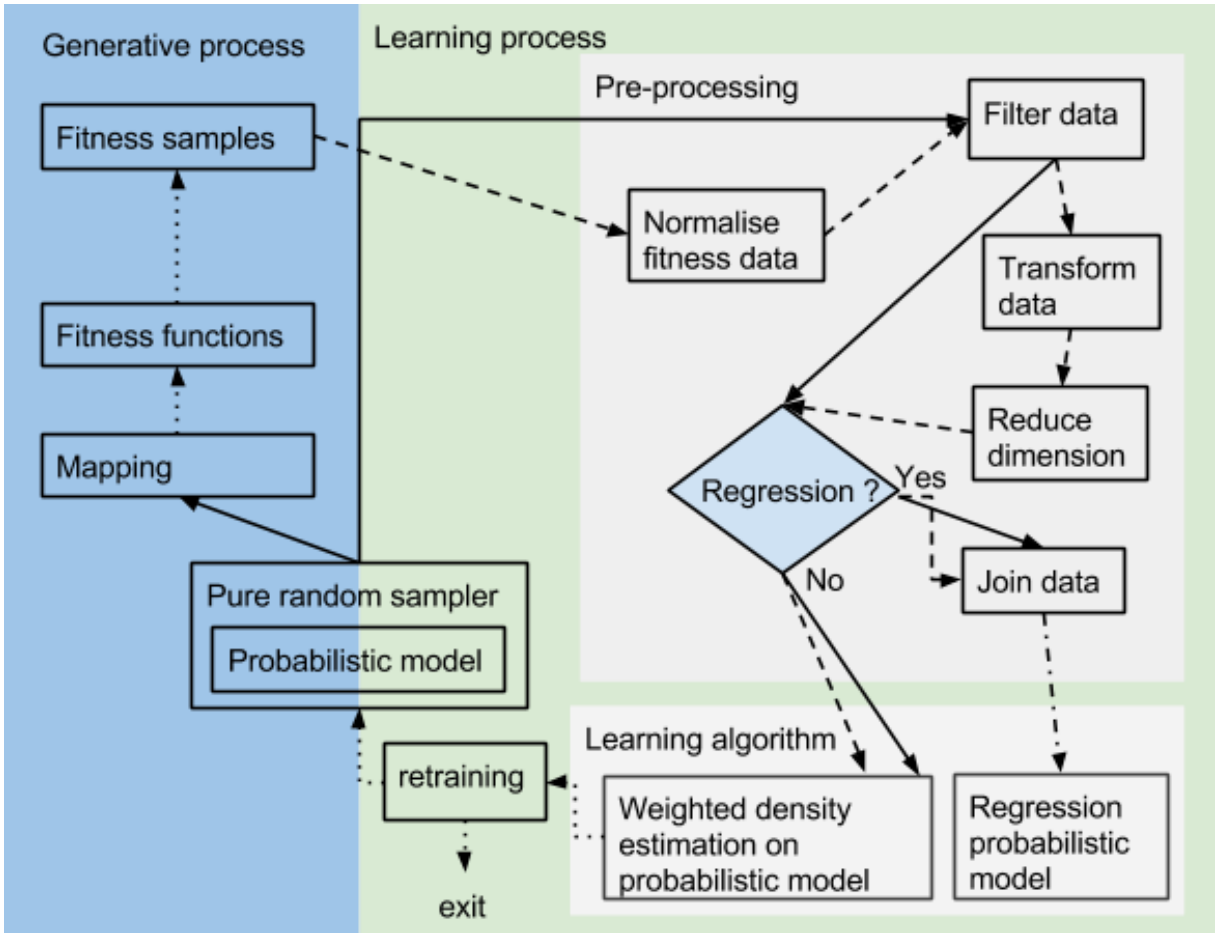


Figure 3.17: Overview of the pre-processing

However, because I use supervised density estimation, I also need to generate the labels for the data. As mentioned in section 3.3.1 these labels are derived from the fitness function values of a layout.

In order to train a GMM I need both the samples and the labels in a fixed data format such that the correct data can later be extracted after sampling from the trained GMM.

Asset data The generation process creates samples from the asset types which will be used to form the training data for learning algorithms. It is not possible to train directly on the hierarchical samples (parent/child) from the generation process, the data needs to be "extracted" from the samples. This extraction will convert the values of variables in a hierarchical scene in a flat vector. Given the set of variables of the parent ($B = var_a^p, var_b^p, ...$) and child nodes ($A = var_a^c, var_b^c, ...$). Then the set of corresponding variable values for a number of n child samples (C')and a parent sample $p'$are respectively defined as $A'_i = val_a^{c_i}, val_b^{c_i}, ... \forall \in C'$ *and* $B' = val_a^{p'}, val_b^{p'}, ...$ Additionally each value is a vector of its own. And a sample row in the asset data is the vector: $B', A'_0, A'_1, ..., A'_n$

The number of children n of which the features are included in the training is smaller than or equal to the actual number of sampled children in the generation process, I will explain later why some child samples are excluded from the training data. n will be derived from a hyper-

parameter in the next loop of the learning process. However, it might not be possible to generate the exact amount children desired for the training data due to a conflict with the generation process.

For example, the user defines a scene layout with 3 to 5 children, and I want to train a GMM with data from 2 children. It is not possible to generate a scene layout with exactly 2 children, but it is possible to train only on a subset of the children. I have to respect the order in the generation process of the children. Because the trained distribution of the 3rd child will not be the same as the 5th child, conceptually it is straightforward to see that the 5th chair will not have the same optimal positions as the 3rd. A subset of n children which still respects the order but which is smaller than the lower bound (n_{small}) is constructed by training on a sliding window of length n over the minimum number of children in the scene. This would result however in $n_{small} + 1 - n$ different subsets. Generating more data from existing data is called data-augmentation in machine learning context. However in order to respect to only have one subset per sample, and avoid some samples to have more influence on the training data, I propose to only take the first subset of the sliding window. I will compare both approaches in the next chapter.

Fitness data In addition to extracting the sampled data, I also need the labels from the sampled fitness values. Given a set of fitness functions $F = f_0, ..., f_k$ and sibling order n with the set of children trained upon $C = s_0, ..., s_n$, I calculate the fitness function values according to their respective relation type. A row in the fitness data is the vector $[f_0(s_0), ..., f_0(s_n), ..., f_k(s_0), ..., f_k(s_n)]$.

Training data transformation After extracting the asset data and calculating the fitness, it goes through a transformation phase which is parameterised to allow different configurations for the transformations. Each different configuration will be evaluated. In this section I will discuss filtering of the data, followed by the polynomial degree of the data from the fitness functions and finish with the dimensionality reduction of the fitness function vector.

Data filtering After I obtain the asset data and fitness data I normalise the fitness values in order to be able to correctly filter them based on absolute values (thresholds between 0 and 1). I propose to filter out samples which have an average fitness function value below a certain threshold. As I use a pure random sampling algorithm and which without filtering would yield samples of both high and low fitness. The user can define this threshold as the lower bound on each fitness function which will be used to filter out the redundant samples. The lower bounds' influence on the learning process depends on the analytical behaviour of a particular fitness function, which I will show in the evaluation chapter.

A second step is to filter out the fitness values which show little to no variance. If a fitness function yields values with low to no variance, i.e. the result is always equal (or close) to a single value. This is undesired for 2 reasons, first a constant value for all samples does not add any information and makes the data redundant , thus the fitness function will not constrain the "a posteriori" distribution of the samples. The second reason is that, for regression, a dimension for which the data has almost no variance leads to numerical instability in the EM algorithm.

Fitness value polynomial order If the fitness data does not show variance, the data can be transformed by increasing its polynomial order. The order controls the steepness in the curve of the function and increases the variance between fitness values. For example if a fitness function' s values stay in the normalised interval $[0.9, 1]$ increasing its order by 2 will result in a range of $[0.9^1, 1^2] = [0.81, 1]$. There is however no standard way of defining a fitness order, which makes it a parameter for the user to tweak, the influence of this parameter will be evaluated in the next chapter. In figure 3.18 I show an example of how the polynomial order affects the fitness function behaviour.

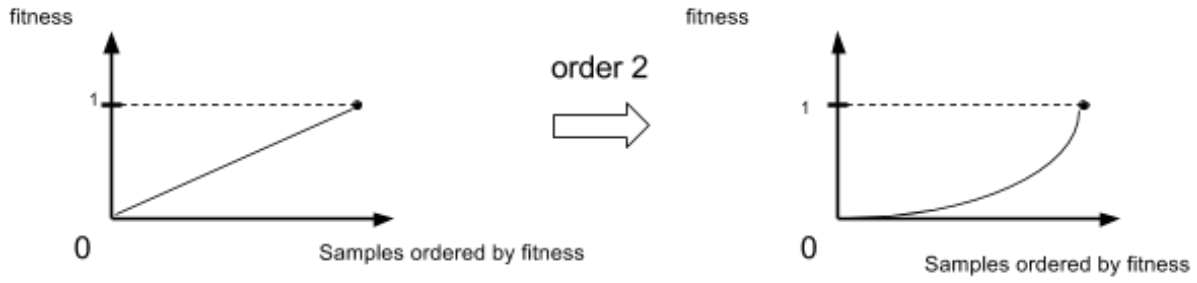


Figure 3.18: Graph for example of applied polynomial order on fitness

Fitness value dimensionality reduction As mentioned in section [learning algorithm] this step is different depending on whether I am using weighted density estimation or probabilistic regression for the supervised density estimation. When using weighted density estimation, the training method only accepts a single dimensional fitness value, the weight of a single sample in the training data. This weight is the normalised likelihood of a sample calculated similar as in Yeh. et al. by simply multiplying the fitness values.

Regression accepts multidimensional input data, increasing the dimensionality of the fitness value vector can cause the "curse of dimensionality" however, which could negatively impact performance. There are 3 dimensionality reduction schemes, each based on the multiplication of the fitness values. I will compare them with each other in the next chapter. The first is identical to the calculation done for the weighted density estimation, multiplying each fitness value, called "single fitness dimension". In the second I reduce the fitness function vector along the dimension of the samples, multiplying only the fitness values of each sample, resulting in a single fitness value for the parent and each child, called " sample fitness dimension" . The last applies no reduction at all and gives us a the original fitness format as fitness vector, called "full fitness dimension" .

Retraining

I elaborate on one last component called retraining. There are two reasons to repeat the training of the probabilistic model, respectively with two different loops called retraining trials and iterations. Each time a certain amount of samples are generated for the training data, this data can, by definition, vary every time it is generated. Therefore the training data might be biased towards particular regions in the search space. The first reason is to avoid the sampling bias in the training data, the trials repeat the training of the same probabilistic model. After n retraining trials I choose the result with the highest fitness from these trial probabilistic models. After updating a probabilistic model, it is possible to re-sample data from this model, recalculate fitness samples for it and retrain its distributions with this data. This could potentially reduce the generators expressive range, because the loss in expressive range in the previous training stage will be carried along in the next and maybe even increased. For the second reason, I argue that in return it is possible to increase the performance of the generator even further, because it adds information about the fitness functions. The loop of repeating the training on an already trained probabilistic model is called retraining iterations.

In diagram 3.19 I give an overview of the hyper-generator focused on the retraining component, details of the other components have been omitted for clarity.

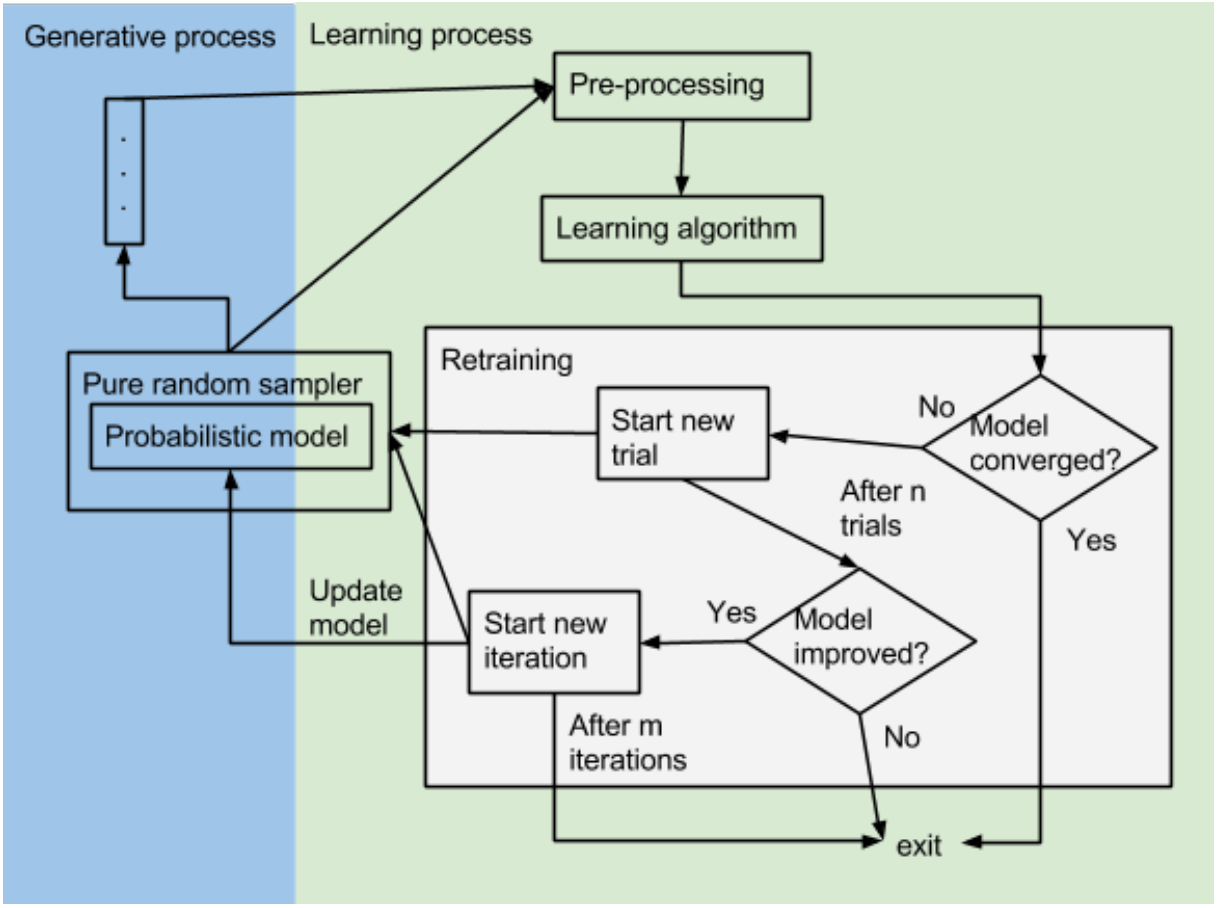


Figure 3.19: Overview of retraining component

Evaluation between training stage After each training stage I will evaluate whether any of the trial probabilistic model has improved over the previous. The metric used to judge whether a model has improved is the average fitness of this model. This empirical metric, defined as a model' s performance, generates X number of samples from the model, calculates each samples fitness and averages it. The fitness values calculated for this metric differ from the fitness values used in the fitness data 3.3.2, where a fitness polynomial order is applied and values are not averaged but multiplied. This is a very simple metric, which does not incorporate the layouts expressiveness. However, I did not find a satisfactory metric for expressiveness that could be evaluated without manual interpretation of the result. Ideally a user should be able to evaluate the model' s expressiveness between iterations. This would result in a mixed-initiative learning process, but is outside the scope of this thesis.

Convergence after training Due to retraining for the purpose of increasing performance, a layout could at one point reach convergence. If the fitness values of a layout are consistently close to the maximum, any additional training would be useless and could lead to singularities in the matrix calculations of the learning algorithms. That is why before a layout is further optimised, it is checked if any of the fitness functions has converged. Convergence is measured using a fixed threshold between retraining steps, if the average fitness surpasses this threshold, retraining is stopped. This threshold is not evaluated, because in all our experiments the average fitness never surpassed the default value of 0.9.

Retraining parameters I briefly summarise the parameters of the retraining process. (1) The number of training iterations and (2) the number of trials. The performance metrics is an empirical metrics, to estimate the model' s average fitness and its the variance between trials, I need to define (3) the number of model evaluations.

3.3.3 Sampling after training process

After updating the probabilistic model in the training process, it no longer has the same hierarchical structure from the user-definition in the generation process. After training, variables can not only be related to a parent variable, but to a sibling parent as well, see section 3.3.1.

The sampling process is the same as the sampling discussed in the generation process 3.2.4. However, with an additional ordering for generating the child samples. Instead of immediately calculating the relative variable values for all children, each child is sampled in order and the sampled density is conditioned on its previous siblings. This ordering does not affect the functional relation between parent and child variables 3.2.1. The relative values for the child variables are calculated, by applying their respective functional relation, after the conditional relation has been enforced.

However, when sampling from the GMM, there is one additional problem that can arise. The range of values sampled from trained distributions can exceed the user defined range for a variable. The figure 3.20 gives an illustrative example, comparing the uniform density for chairs positions with GMM trained to estimate this density. In order to respect the user defined ranges,

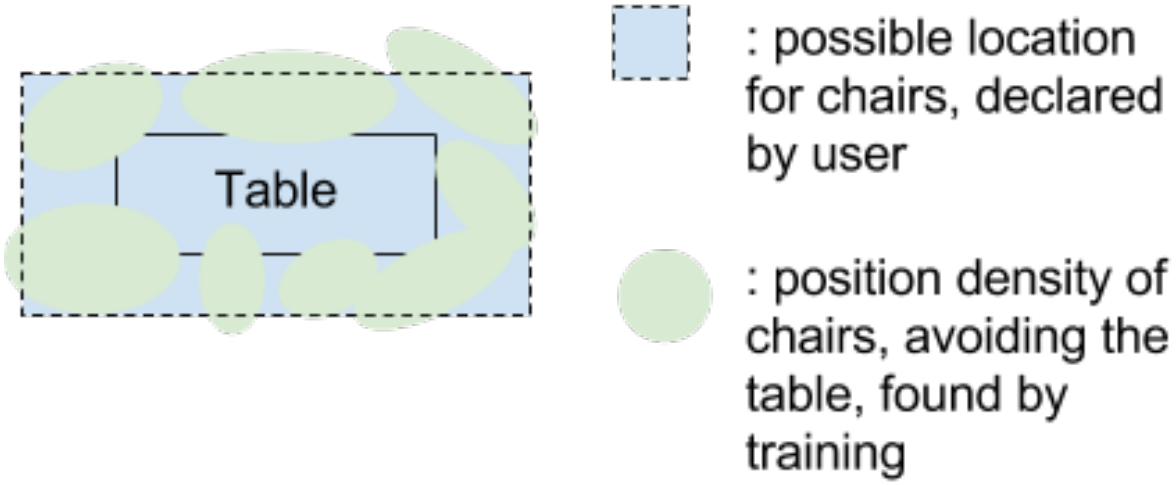


Figure 3.20: Example of trained distribution exceeding user-defined distribution

a rejection sampling method is employed. This method is still purely random. The rejection sampling simply samples from the distribution until a value within range is obtained. To avoid an endless loop there is a maximum number of tries. But because the models is trained on data within this range, the range of the distribution is always relatively close to the actual bounds.

Figure 3.21 is an adapted version of the recursive sampling in figure 3.12. In the adapted version the red arrow not only visualises the execution of the functional relation between parent and child node, but the conditional relation between parent and child as well. A sample contains the values of the variables in a sampled node. The blue arrow denotes the conditional relation

between the variables of the sampled child and its previously sampled siblings. For a detailed

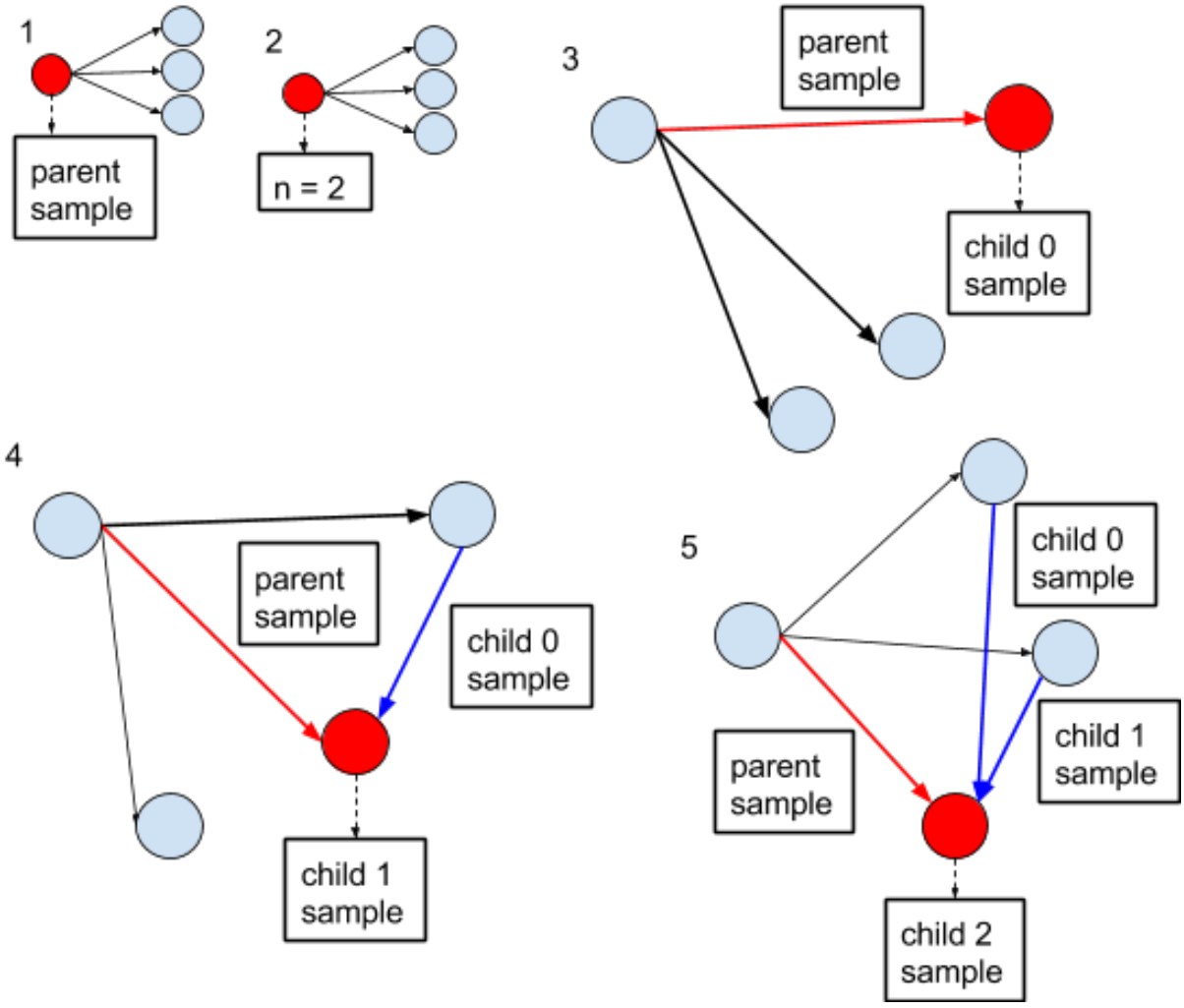


Figure 3.21: Example of trained distribution exceeding user-defined distribution

explanation on how the variables in a child node are conditioned on its previous sibling and its parent I refer to the section 3.3.1.

Chapter 4

Evaluation and discussion

The first section elaborates on the content I will generate, namely scene layouts. The second section will discuss the results obtained through experiments

4.1 Scene layout generation

As mentioned in the previous chapter, I will take a scene layout in architectural context as my use-case for evaluating the methodology. Scene layouts are a suitable type of content because it has both bottom-up and top-down properties. The hierarchical context implies the semantics of the model variables and its fitness functions. Additionally, I tried to partition the fitness functions based on their analytical characteristics including continuity and multi-modality. In a scene layout the placement of the objects is a very important part of the generating procedure and distinctly different in terms of algorithm from asset creation. To generate each object's shape, size, rotation and position I declare a node for each object that contains these variables. An object's shape is generated as a polygon and is defined as a collection of points. Shape, size, rotation and position can all be stochastic variables. I repeat that stochastic variables have random values and deterministic variables fixed values. However, they will not all be stochastic at the same time when evaluating. This reduces the amount of non-deterministic variables in the content in order to have more reliable results. In the next section I will discuss fitness functions for scene layouts and how they are used during evaluation.

4.1.1 Fitness function

The constraints of the scene, defined as fitness functions, are inspired by the works of Yeh et al. and Merrell et al., who both discuss scene generation with constraints. The semantics have no influence on the learning process but identifying and using the more common constraints in a particular domain improves the realism and quality of the evaluation. There are an infinite amount of possible fitness functions and each fitness function needs a search space in which it is possible to evaluate them. An exhaustive analysis of the possible influence of each function on the learning process is out of scope for this thesis. I will however evaluate the most commonly used constraints and compare them, based on characteristics necessary to distinguish them in terms of behaviour relative to their input. In this section I will elaborate on the fitness characteristics and define each fitness function separately.

Fitness characteristics

As mentioned in related work, in order to calculate fitness functions I need to map the search space, the stochastic variables of the scene layout, onto a representation that can be evaluated by the fitness function. In my methodology I mentioned that I will not be evaluating the influence of different mappings, therefore I restricted the evaluation to two possible mappings. The first trivial mapping, is a copy of the variables, $m(a) = a$. For example to calculate the distance between positions, I only need the position variables without any additional transformation. The second mapping, maps the shape, size, rotation and position variable to a polygon representation of the scene. For example to calculate the alignment angle between two polygons. A fitness function which is based on goniometric functions $(cos, sin,)$. Another type of functions, which also requires a polygon mapping are the geometrical functions, for example the overlapping surface area of two polygons. I will distinguish fitness functions along three dimensions:

- Mapping
  - none
  - polygon
- Hierarchical relation
  - parent-child
  - siblings
  - absolute
- Analytic family
  - distance based
  - goniometric
  - geometric

The table 4.1.1 lists all fitness functions which will be used during evaluation and positions them on the respective axes.

	Hierarchical relation	Mapping	Analytic family
Minimum polygon overlap	pairwise	polygon	geometric
Target surface ratio	absolute	polygon	geometric
target distance	pairwise	none	distance based
Minimum centroid difference	absolute	polygon	geometric/distance based
Closest side alignment	pairwise	polygon	goniometric/geometric

Table 4.1: Fitness characteristics

I will elaborate on the calculation and semantics of each fitness function in the coming subsections. With regards to the hierarchical relations, I refer the reader to [section in methodology] for an elaborate explanation.

Minimum polygon overlap and Target surface ratio

The minimum polygon overlap and target surface ratio are closely related. The minimum polygon overlap is a fitness function which measures the overlapping area of two polygons. Because I want to avoid collision between polygons, the fitness function additionally negates the surface. This will have as result that the fitness function, and therefore the polygon overlap, will be minimised. The target surface ratio will also calculate the surface of overlapping polygons.However, for two or more polygons because it is an absolute fitness function in contrast to the minimum polygon overlap which is a pairwise fitness function. Instead of minimizing this surface, I want the fitness function to target a specific ratio of overlapping surface. This means that the overlapping surface will be normalised, no overlap equals 0 and complete overlap 1, which results in the overlap ratio. The last step is to have the function target a specific ratio, or in other words minimize the difference between the actual ratio and the target ratio. If for example a set of plates has to cover 80 % of the surface of a table, the fitness function would be the normalised area of the combined surface of the plates covering the table's surface, subtracted by 0.8. The mathematical equations of $f_{po}()$ and $f_{tr}()$ are given in 4.14.2, respectively for minimum polygon overlap and target surface ratio. In the equations s stands for sample, S_i for the i'th child sample, s_p for the parent sample, $pol(s)$ for the polygon mapping of a sample, $A(p)$ calculate the surface area of polygon p.

$$f_{po}(s, s') = -A(pol(s) \cap pol(s')) \tag{4.1}$$

$$f_{tr}(r_t, p, s_0, .., s_n) = r_t - A(\cup_{i=0}^{n}(pol(s_i)) \cap (pol(s_p)))/A(pol(s_p)) \tag{4.2}$$

4.1.2 Target distance

The target distance function has been mentioned in the methodology before and is shown equation 4.3. Where $d = sqrt((s.pos_x - s.pos_x)^2 + (s.pos_y - s.pos_y)^2)$, m can vary and $Mis\infty$.

$$t(d, m, M, \alpha) = \begin{cases} (\frac{d}{m})^{\alpha} & d < m \\ 1 & m \leq d \leq M \\ (\frac{m}{d})^{\alpha} & d > m \end{cases} \tag{4.3}$$

4.1.3 Centroid difference

In Merrel et al. they argue that visual balance is the most widely known principle of visual composition. The visual balance in a room is a constraint with absolute relation between objects. To measure a degree of visual balance the distance between the centroid of the room and the centroid of the objects in the room is calculated. This centroid is derived as the average position of the objects weighted by their respective surface in the room. In figure 4.1 the visual balance of a room is visualised with the distance (d in the figure) between the centroid of the room (blue cross) and the weighted mean position of the objects in it (red cross), weighted by their respective surface. The equation is given in 4.4, where R is the room, S the collection of samples in the room, $pos(s)$ the position of a sample and $c(R)$ the room's centroid.

$$m_{vb}(R, S) = -\left\| \frac{\sum_{s \in S} A(pol(s)) \cdot pos(s)}{\sum_{s \in S} A(pol(s))} - c(R) \right\| \tag{4.4}$$

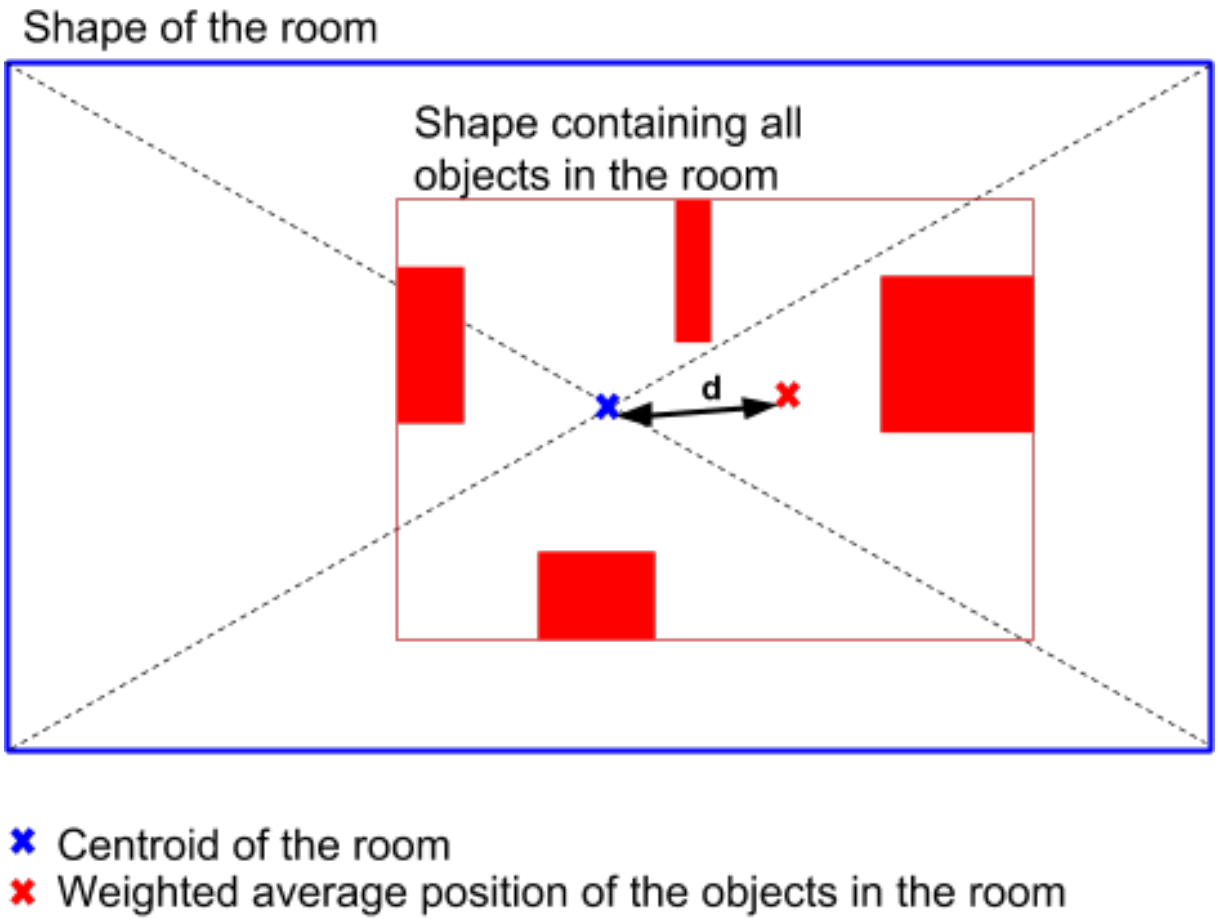


Figure 4.1: Visualisation of visual balance in a room by centroid difference

4.1.4 Closest side alignment

In Merrel et al. they state three reasons why objects in a scene should be aligned to a certain angle. If the objects can be sat upon, there are two possible constraints concerning the object's angle. Firstly, they could face each other in order to support conversation. Secondly they could focus on a single focal point in the room, for example on a piece of art. Thirdly, for objects in general, in order to create an aesthetically pleasing object arrangement, objects should be placed perpendicular or parallel to each other. The closest side alignment is the fitness function for orientation of objects in the room relative to each other. With this function I want to encourage two objects to be parallel or perpendicular. The angle is calculated between the closest side of their two respective polygons. Even though most objects can be represented by a polygon with straight angles, including a chair, cupboard and a carpet. There exists furniture with irregularly shaped polygons as representation, for example a table in the form of a pentagon. If an object has non-straight angles in its shape, simply aligning the objectv́tick s rotation will not suffice. I will need to align the sides of their respective polygons. The fitness is analytically defined by equation 4.5, where $cs(pol, pol')$ returns the respective closest sides of two polygons and $angle(l, l')$ the angle between two sides. The angle is multiplied by 4 in the calculation of the cosine in order to have high fitness for side angles with $0, 90, 180$ or 270 degrees.

$$f_{csa}(s, s') = (1 + cos(angle(cs(pol(s), pol(s'))) \cdot 4)/2 \tag{4.5}$$

In figure 4.2 I visualise an example of the closest side angle polygons, between the red polygons and the blue polygon. For a detailed explanation, terminology and discussion of constraints for

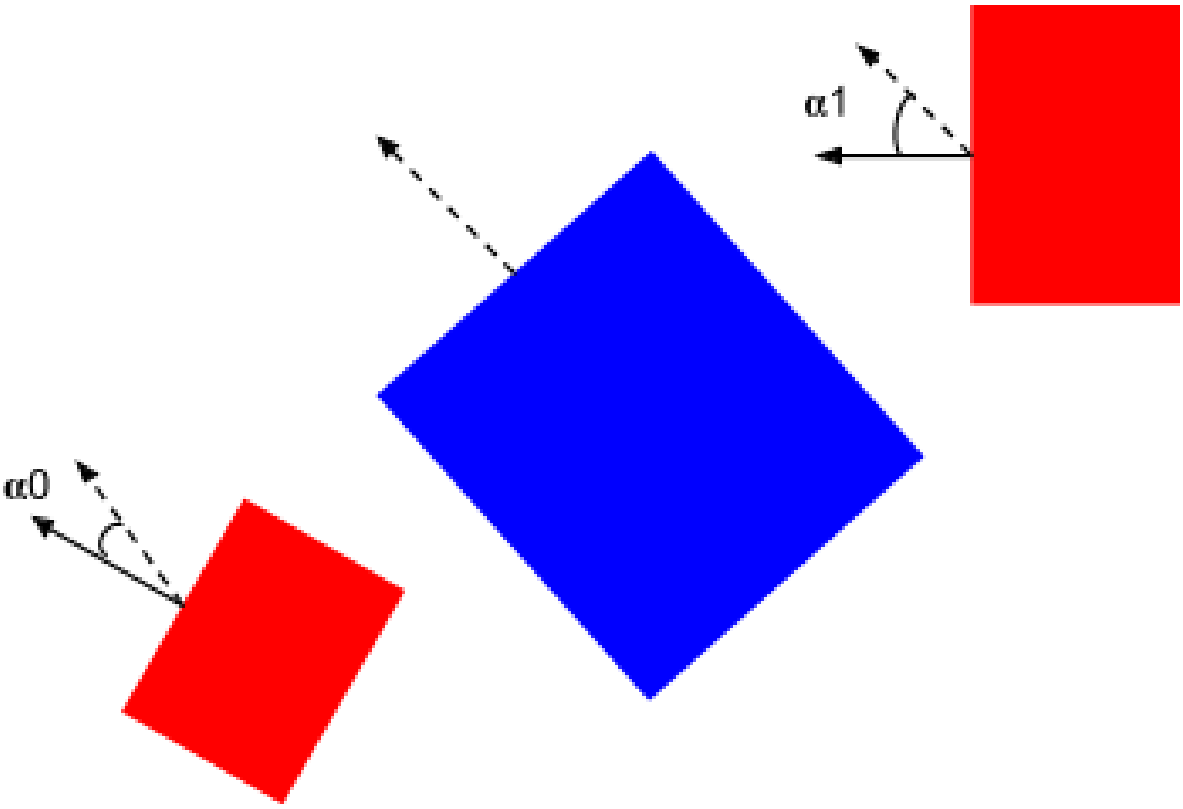


Figure 4.2: Example of closest side angle

furniture layout, called criteria in the paper, the reader is referred to [73].

4.2 Experiments

In this section I will discuss the experiments I have performed, their setup and results. First, I will discuss the experiment setup in general. The learning algorithm has a handful of parameters and, depending on the experiment, these will be set to a fixed value. Secondly, I will do an elaborate discussion on the evaluation of expressiveness in the content. Thirdly, the parameters will be separately evaluated over a range of possible values and I will discuss their respective results. In the final section I briefly summarise how well each aforementioned fitness function has been optimised.

4.2.1 General experiment setup

In list 4.2.1, I give the default values for the parameters of the learning method. Unless otherwise stated these values will be used in all the experiments below. For an elaborate explanation of

each parameter and the possible range of values they can have I refer the reader to the section the learning process in the methodology. The parameter are grouped per component of the methodology they belong to.

- Model hierarchy parameters
  - sibling order list = [0, 1, 2, 3, 4, 4, 4]
  - non marginalisation list = [False, False, True, False, True]
- Data generation parameters
  - number of data samples = 500
  - Poisson disk sampling = False
- GMM parameters
  - minimum covariance = 0.02
  - number of components = 30
  - Regression = false
- Retraining parameters
  - number of trials = 5
  - number of iterations = 1
  - number of samples for evaluations = 500
- EM parameters
  - tolerance = $1e-3$
  - number of initialisations = 5
  - number of iterations = 100

4.2.2 Expressiveness

Expressiveness is a quality which is very hard to measure [32]. That is why to evaluate expressiveness in our methodology we will try out several different approaches based on visualisation. However due the high dimensionality of the problem most of these require a more ad-hoc approach on low-dimensional use-cases (< 3), without taking the fitness functions into account. As aforementioned the content for which I evaluate expressiveness has a 1 on 1 mapping for each asset instance of the content (either a copy of the variables or a polygon mapping). Therefore, I will evaluate the expressiveness of each asset instance in the content as the expressiveness of their respective variables. The expressiveness of the variables will be visualised as the distributions from which they are sampled. As mentioned in the methodology these distributions are initially uniform and consecutively trained on the data using a GMM as underlying model. Also mentioned in the methodology is that a uniform distribution has the maximum expressiveness, it has no bias towards any particular value. I will visualise the trained distributions, no longer uniform, in order to show their bias. This bias will be one metric of expressiveness. The second metric for expressiveness is called expressiveness coverage. Similarly to visualising the bias I use a the distribution of the variables to evaluate this metric. However, instead of comparing the bias between uniform and trained distribution I compare the area within the user-defined bounds of the variable, each distribution covers. A uniform distribution has an optimal expressiveness for this metric, each value within these bounds has equal probability to be sampled. However, when sampling with a trained distribution, it could be that some values are excluded

and therefore some areas within the bounds UNcovered. Expressive range for game levels is extensively discussed in Smith et al. [16]. The paper presents a more rigorous approach to analyzing procedural content generators by classifying the style and variety of game levels that can be generated. In order to fully capture the range of content that can be created and analyse how this expressive range changes for different fitness functions or generation parameters. They visualise expressive range with 2D heat maps on high level features of the levels. These features represent a desirable property of the content reduced to a single dimension. The heatmaps have one feature on each axis and visualise the distribution of a certain amount of generated content over these features. An example is given in figure 4.3 originating from Smith et al. The visualised features are the leniency and linearity of a level. This exact approach is not applicable to

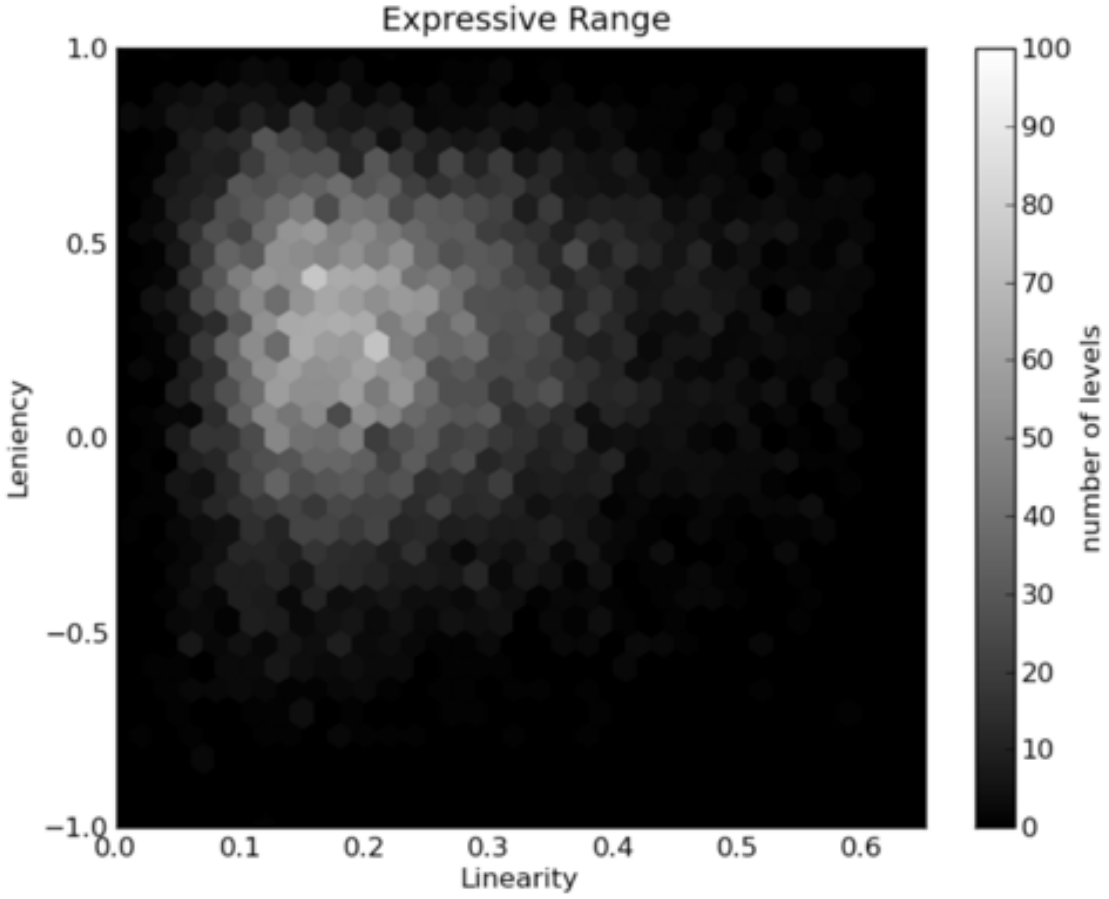


Figure 4.3: Expressive range of a game level, visualised with heat map

my use-case because I do not need to define high level features. Each random variables in my use-case has a semantic meaning and thus each of these variables should have an as large variety as possible. The random variables are defined by density functions, therefore instead of variety I speak of the variance and bias of their density. This allows me to measure expressiveness immediately on the variables, without further transformation. In the next section I will analyse the expressive range of my methodology. First in terms of variance and how much of the variance covers actually desired content. Secondly in terms of bias towards particular values.

Expressive range coverage

For one use-case I will do an exhaustive expressiveness analysis. I will restrict variables in the use-case to two dimensions, namely the position of a single object in x and y. This use-case also visualises the validity of other concepts from the methodology; simultaneous parent-child and sibling conditioning, optimising multiple fitness functions' and marginalisation for re-use of a trained model. I will explain in detail with the respective figure how these concepts can be noticed. Additional configuration of the generation process in the use-case is; a model with one parent, three child shapes and two fitness functions which influence the optimal child object position. Namely polygon overlap between parent and child objects, as well as target distance between siblings. The parent has a variable as well; its shape. And the child position will be conditioned on both its sibling position as well as the parents shape.

Experiment setup For the evaluation the number of samples used as training data is set to 5000, in order to avoid any influence on the result due to insufficient training samples. In the figures which visualise the expressive range the ellipses are a two dimensional representation of the gaussian components in the trained GMM. Each outer ellipse contains the two order confidence level (95%) of the Gaussian, called Gaussian surface area (GSA). The performance will be measured using terminology from statistics, specifically sensitivity and specificity. Specificity or true positive rate, gives the portion of the desired content which is covered by the GSA, the sensitivity or true negative rate measures the portion of undesired content which is outside the GSA. The parameters for the trained model are given in the general experiment setup 4.2.1 with some changes in list 4.2.2.

- Model hierarchy parameters
  - sibling order list = [0, 1, 1, 2, 2]
  - non marginalisation list = [False, True, True]
- GMM parameters
  - number of components = 15

Expressiveness results In the figures 4.2.24.2.2 the blue and red shape are respectively the parent and the child. The visualised distributions in figure 4.2.2 and 4.2.2 are respectively conditioned on the parent only and both the parent and the child. Thus the green area shows the most probable location of the next sampled child. The first approach to visualise the expressiveness will compare heat-maps between naive rejection sampling, also called generate and test, and rejection sampling with our method. This empirical approach will be done with 10000 iterations.

The figure 4.2.2 illustrates what the result of an "ideal" sampler would be. The sampled points are uniformly distributed over the search space in which each sample has maximum fitness. I repeat, no overlap with the parent and a distance of 1 from the sibling. From the figure 4.2.2 it is clear that the trained model was able to capture the coarse shape of this optimal search space. In order to evaluate the trained models performance I calculated the average rejection rate of 10000 samples which was approximately 70% for the naive sampler and 86% for the sampler with trained model. This might seems like a marginal profit, however the speed-up gained from training is usable every time the model is sampled. This means that every time content generator is used, it will be 22% faster.

A second approach, is to visualise the expressiveness by assuming that the surface area of the Gaussian represents the models complete possible sample space, and not only 95%. However, in order to have an accurate estimate of the surface covered by the sample, some components

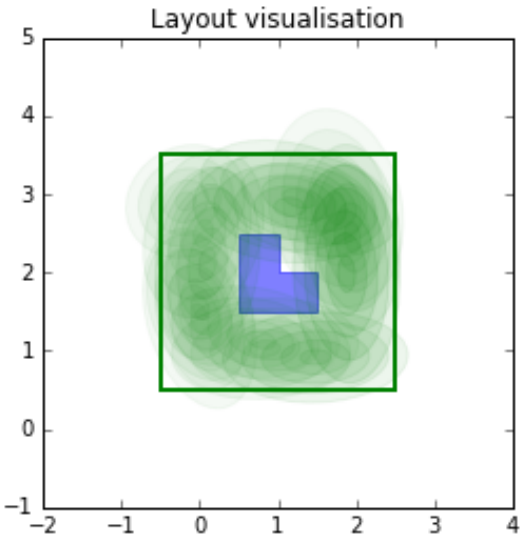


Figure 4.4: Distribution for child position not overlapping its parent

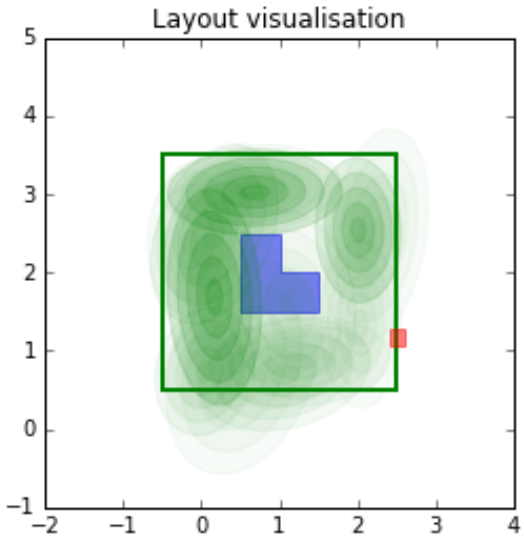


Figure 4.5: Distribution for second child position not overlapping parent its and with distance 1 from the first child

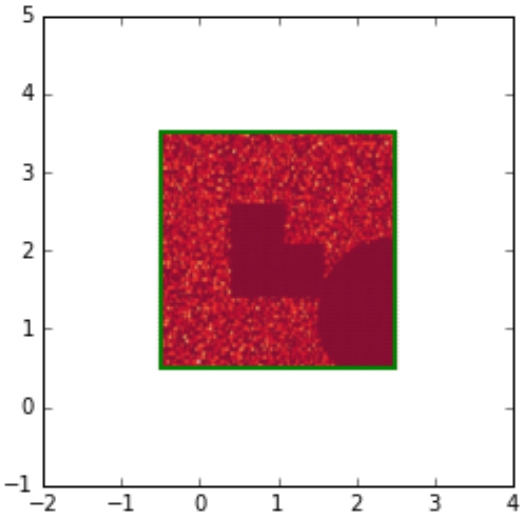


Figure 4.6: Heat-map for naive sampling

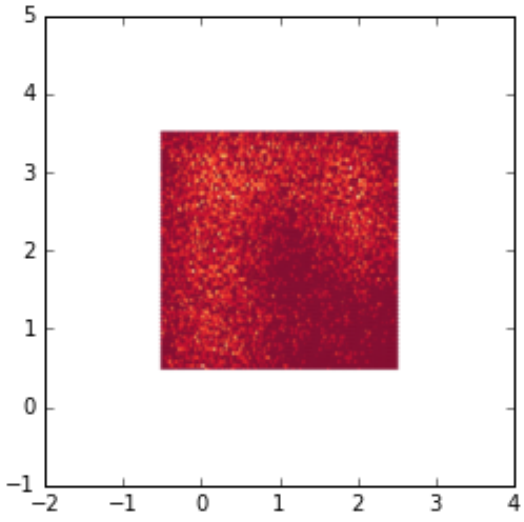


Figure 4.7: Heat-map for sampling from trained model

are excluded from analysis. As aforementioned, each component has a relative weight, which indicates the probability of selecting in the sampling process of the model. If this weight is below 2 times the uniform weight ($2/n_components$) than it is omitted from the visualisation. This value has been derived from initial testing before the experiments. In the next paragraph I will discuss a downside of this method.

The figure 4.8 is color coded. Green is the true positive area, the area covered by both the trained model and the optimal search space. Red is the false negative area, which is not covered by the trained model but which contains desired samples. Cyan stands for the area covered by the trained model but outside the original search space of the naive sampler. Samples from this area are easily filtered, but do lead to a small amount of wasted iterations when sampling. The exact shapes allow me to calculate an analytical score for the expressiveness. Shown in table 4.2.2 with the performance in terms of sensitivity and specificity.

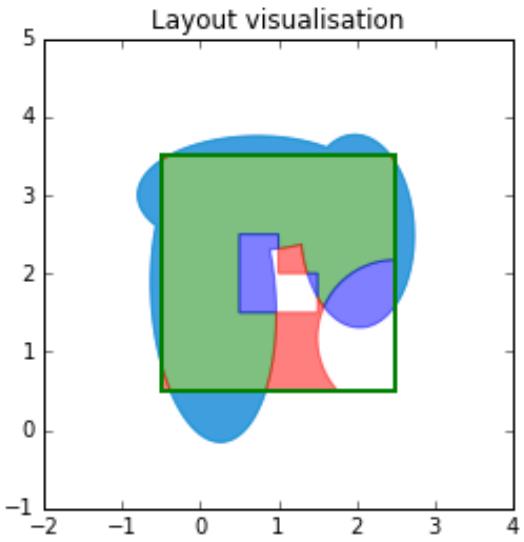


Figure 4.8: Expressive range example, with statistical color coding

In addition to the previous conclusions regarding the expressiveness and performance of a trained model. I show figure 4.9 which visualises the same model conditioned on a different parent shape, in order to generalise that these conclusions still hold when the parent variable changes. This illustrates that the same model is usable to optimise sampling over a range of possible parent shapes.

When comparing the heat-map in figure 4.2.2 (bottom right) with the corresponding color coded area (top right) it can be seen that a big part of the Gaussian surface areas has been excluded from the visualisation. This is due to the method of measuring expressiveness by

		Predicted condition	
	Total population	Predicted condition positive	Predicted condition negative
True condition	Condition positive	True positive: 89%	False negative: 11%
	Condition negative	False positive: 15%	True negative: 85%

Table 4.2: Sensitivity and specificity of analytical expressive range

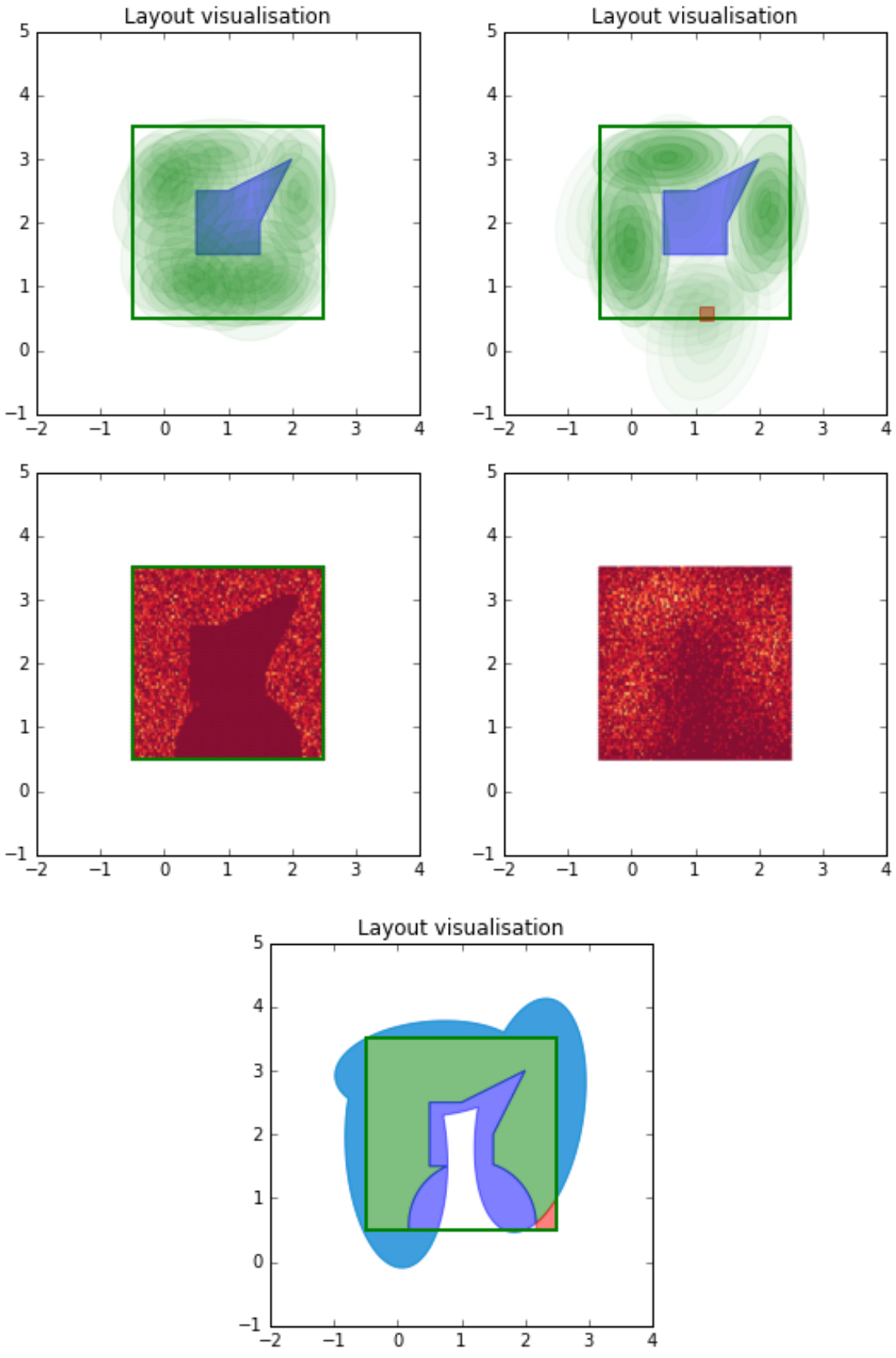


Figure 4.9: Complete expressive range coverage visualisation

filtering component based on hard limits on their weights. It is not perfectly stable and can give unreliable results in some edge cases ($< 5\%$). Which I checked manually over 40 cases. Figure 4.2.2 (below, combine in single figure) is an example of such an edge case from these 5%. In

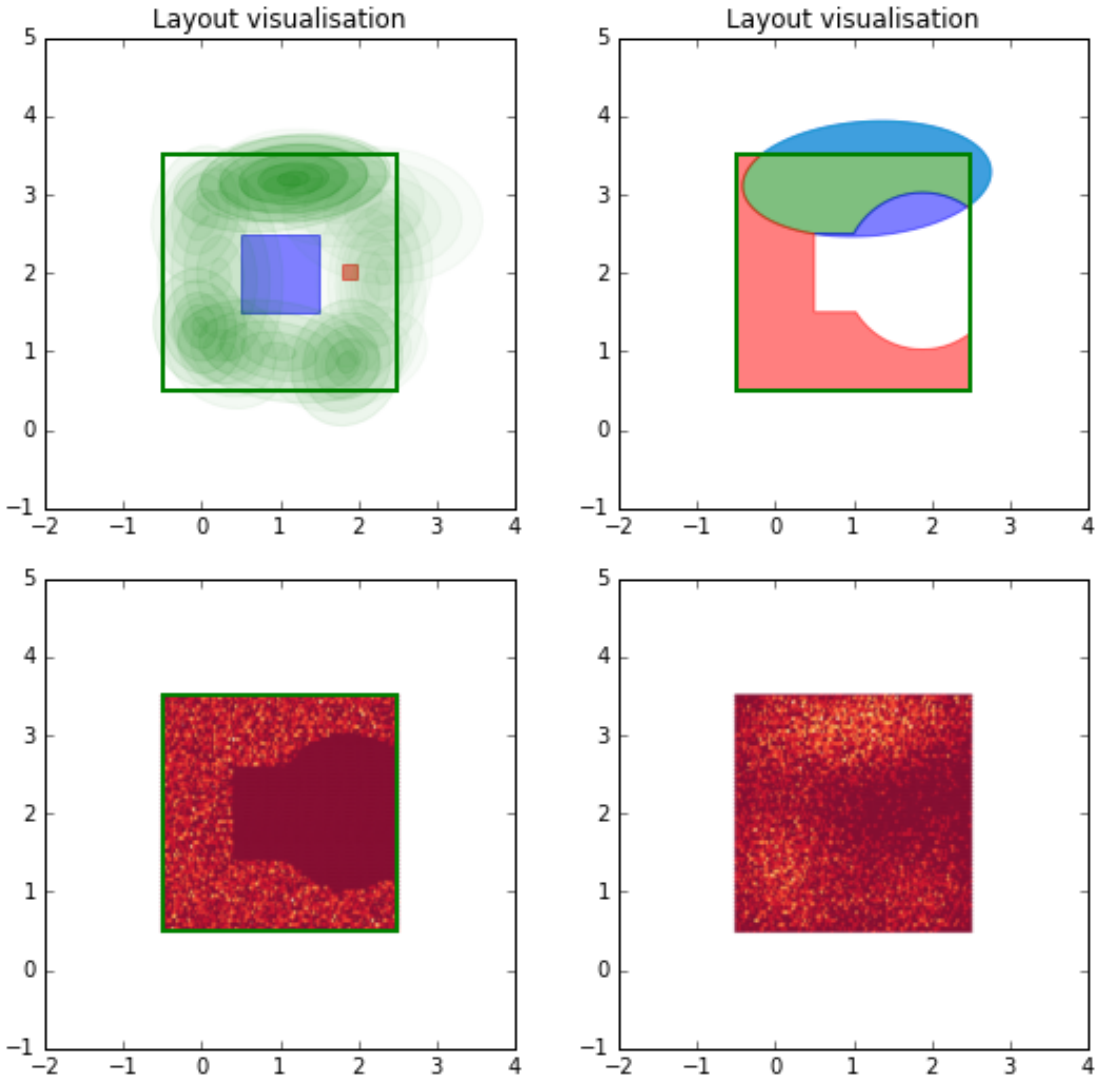


Figure 4.10: Edge case where expressiveness metric fails

previous experiment the trained model only generated two children. In figure 4.2.2 I additionally visualise the generation of 5 children with the trained model. This shows that a trained model is able to capture some dependencies between a larger amount of children and the parent.

Expressive range bias

In this section I discuss and evaluate the expressive range in terms of bias towards a particular value. First I will elaborate on the metric and parameters of the experiment.

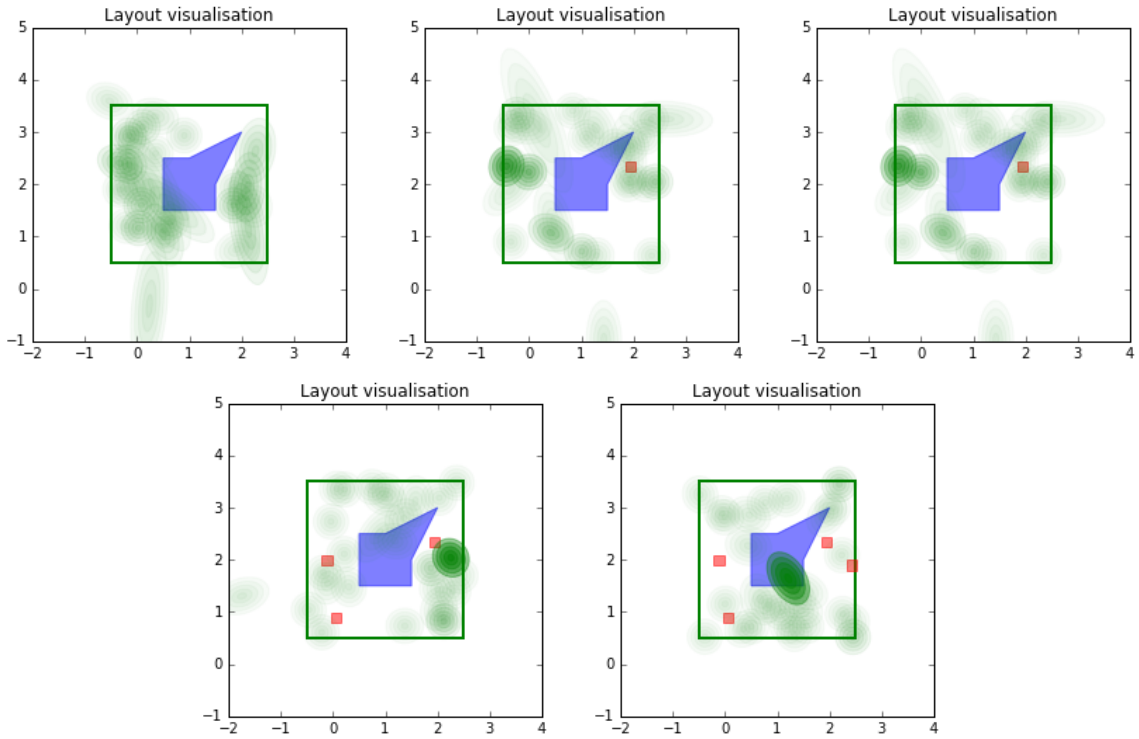


Figure 4.11: Distribution of trained model to generate 5 children

Experiment setup In this experiment I will introduce another way of visualising the expressiveness without reducing its dimensionality, which shows the bias of a random variable. I will do this by marginalising the variables from the joint distribution that models them. However, this throws away the information on the covariance between variables. This metric is however much more scalable compared to the previous one. In the figures of this experiment the blue line visualises the trained density function, the vertical green line the range of the designed search space and the horizontal green line the uniform distribution before training. In terms of performance I now use the gain in normalised fitness values per fitness function after training. The number of training samples are set to 1000. The parameters for the GMM are the same as in the previous experiment and given in list 4.2.2.

Performance results

- Fitness before
  - parent fitness = 0.69
  - child fitness = 0.57
- Fitness after
  - parent fitness = 0.78
  - child fitness = 0.69

In list 4.2.2 the gain can be calculated for the parent and child fitness, the trained model gained respectively 8% and 12% on average for parent minimum polygon overlap and sibling minimum target distance. From this I conclude that it is possible to increase the fitness of some fitness function between both the parent and a child node and of some fitness between the siblings.

Expressiveness results In figure 4.2.2 the graphs with marginalised variables show that the trained distributions still cover the full range of the designed search space. Additionally can be seen that the bias towards particular values is limited, the peaks in the trained density are close to the uniform distribution.

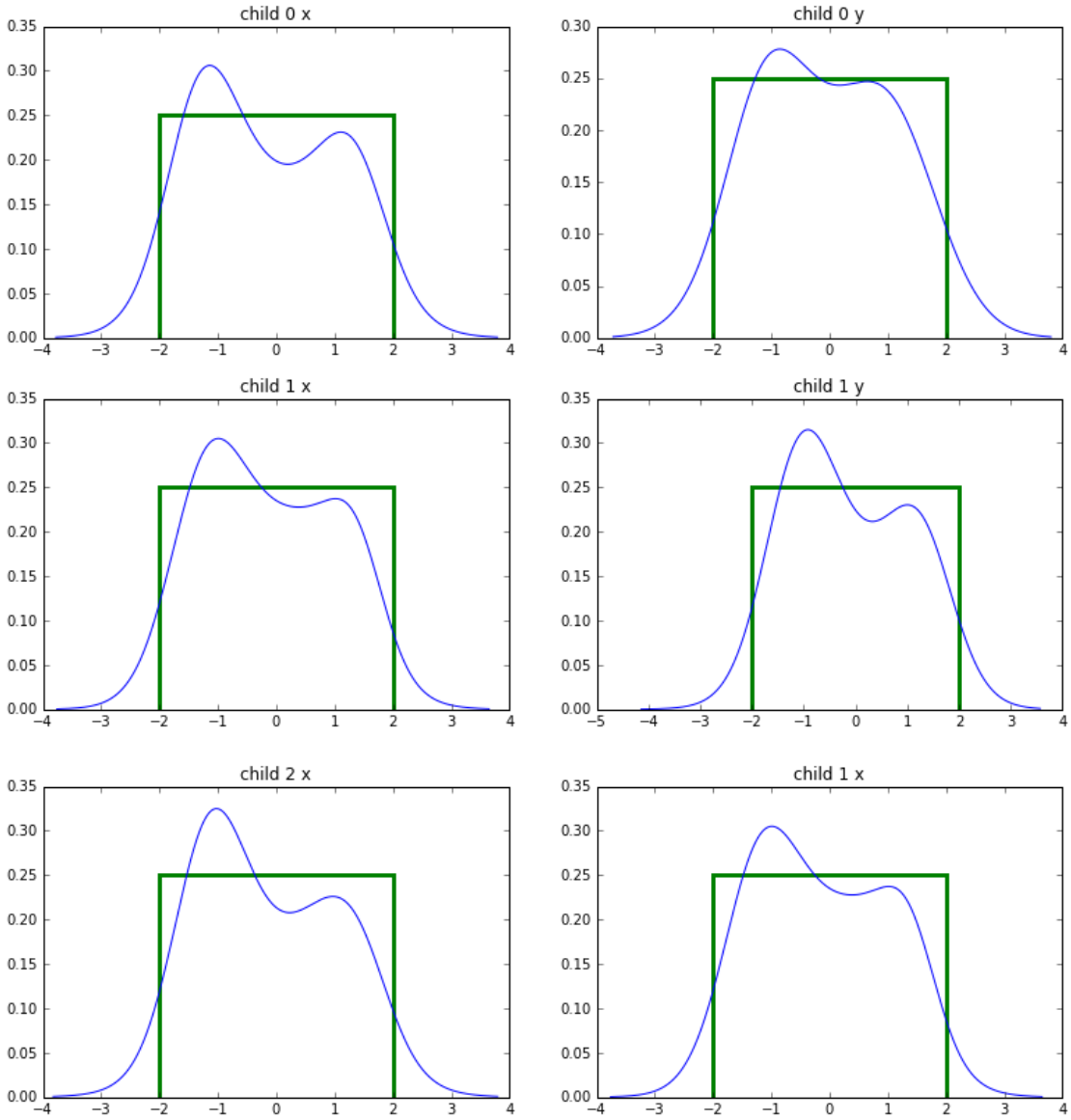


Figure 4.12: Bias in expressive range of trained model, visualised by marginalised distributions

4.2.3 Methodology Parameters

In the following test I focus each time on one or two parameters, let them vary between a range, and analyse the results in order to be able to discuss their influence on the metrics. The metrics are expressiveness, performance and computation time. In order to have reproducible test results I will always save the complete parameter configuration of the test. However, I will only show and discuss the results and parameters relevant for the property I want to evaluate

in each subsection. All files containing the results are indexed and the index is referenced when respective results are discussed.

In the first subsection I elaborate on the experiment setup. Secondly, I will separately discuss the results from experiments which each vary only one or two parameters.

Experiment setup

This subsection discusses the metrics which are used, the scene layouts used for evaluation and some preliminary remarks on the result of expressiveness. The values for the parameters are given in the general experiment setup unless stated otherwise.

Metrics Expressiveness will be visualised by marginalising each trained variable separately, as done in test 2. The difference in the figures of this experiment is that the density is green and the user-defined range of the density red. Due to the more complex fitness functions it is no longer feasible to use the additional visualisation as in test 1 (with exact surface calculation and color coding). However, the visualisation is still useful to evaluate a trained distribution by comparing their Gaussian surface area (GSA) 4.2.2. This means that I will compare different parameter values with their respective distribution for the layouts stochastic variables. This metric is somewhat subjective because it has to measured by manually comparing two visualisation of GSA. Additionally, GSA is most informative when it used for two dimensional variables. For this reason and another mentioned in paragraph 4.2.3 I will compare expressiveness for the position variable, elaborated upon in the next subsection.I will not include every figure of every variable but if an anomaly or trend arises it will be reported. I will rather focus on providing an example of the most common cases and provide highlights of anomalies. The performance of a trained model will be measured as the difference in average fitness before and after training, called average fitness gain (AFG). The fitness gain is not a completely deterministic metric and has no unit. It is however sufficient to compare the performance between different parameter settings. The fitness average is not a deterministic measure for three reasons. The first is the bias in training data for the learning algorithm could result in a different fitness average for the same parameters. The second is the that the calculation of the fitness average of a model (both the initial and trained model) is done by calculating the fitness on other samples from these models, which can be biased as well. The third reason is due to the EM learning algorithm which can get stuck in local optima. The variance between different runs due to these bias will be evaluated in 4.2.3 from which I will deduce a lower bound on the significance of this metrics. The computation time is measured as the amount of seconds to execute the learning method. This is a coarse and non-deterministic metric because the amount of iterations in the learning algorithm (EM) can vary between consecutive runs. In order to reduce the variance for both the performance and computation time caused by the learning algorithm. I will average the results from these metrics over 5 layouts, elaborated in the next section, for which a probabilistic model will be trained in each experiment.

Training models Each fitness function mentioned in the table 4.1.1 will be evaluated separately. I designed layouts, in the form of probabilistic models, for these fitness functions such that is possible to generate content from these layouts which can have high fitness values with a reasonable amount of iterations. This is necessary in order to be able to use the data extracted from these samples to train the GMM. To improve the generality and reduce variance of the results each metric will be averaged over all models. These metrics will be used to compare between different parameter configurations. In order to deduce each parameter's influence on the metrics. The layouts I will use to evaluate with each fitness functions, are visualised below. These layouts are stochastic and a full description of their parameters is given in the appendix. But in order to give the reader an idea, I give 2 example samples of each layout in figures 4.2.3

4.2.3 4.2.3. In the figures visualising the layout, the large blue shape is the parent and the small red shapes are the children. In the first layout, of which 2 example samples are given in figure 4.2.3, the children can change both position and rotation. This layout will be used to evaluate the fitness functions minimum polygon overlap and target distance with respective hierarchical fitness relation, see the section on fitness functions in the methodology; pairwise between a parent and child node and pairwise between siblings. In the second layout, examples visualised in

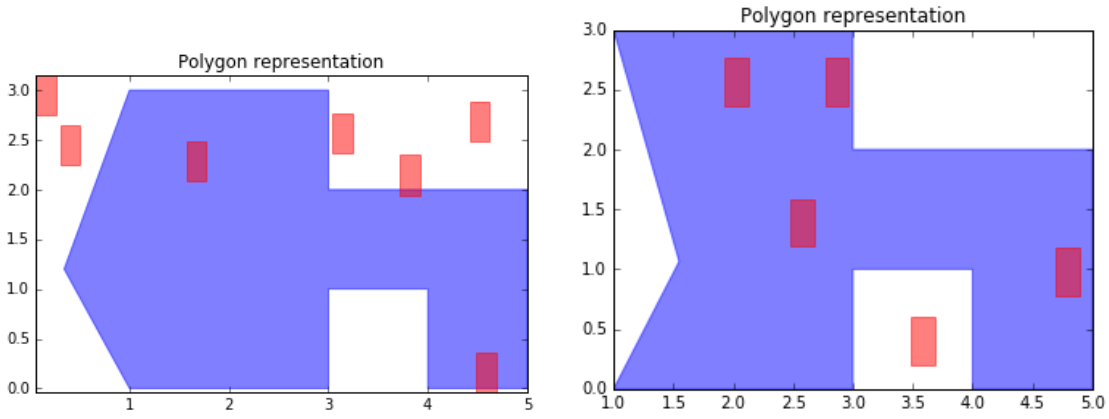


Figure 4.13: Two example samples of the first layout

figure 4.2.3, the children can change position and rotation. This layout will be used to evaluate the closest side alignment with fitness relation pairwise siblings. The third layout, figure 4.2.3,

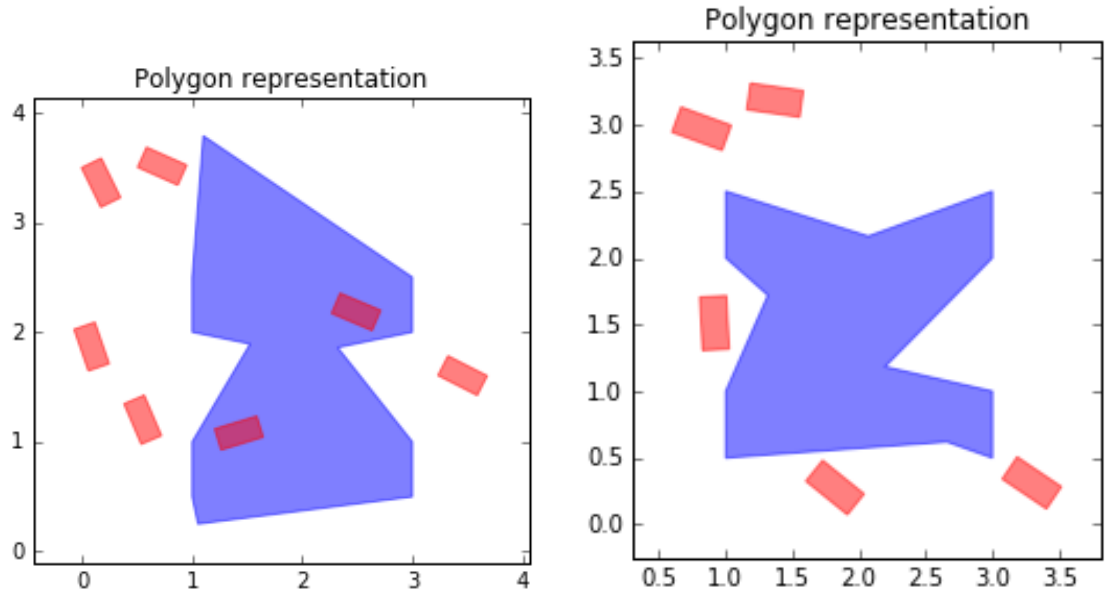


Figure 4.14: Two example samples of the second layout

has children with variable position and size. This layout will be used to evaluate the surface ratio target and minimum centroid difference with both absolute fitness relation.

Naming convention For the detailed results, saved to aforementioned files, I give a summary of the naming and indexing I used for the models, in order to get results for each fitness function

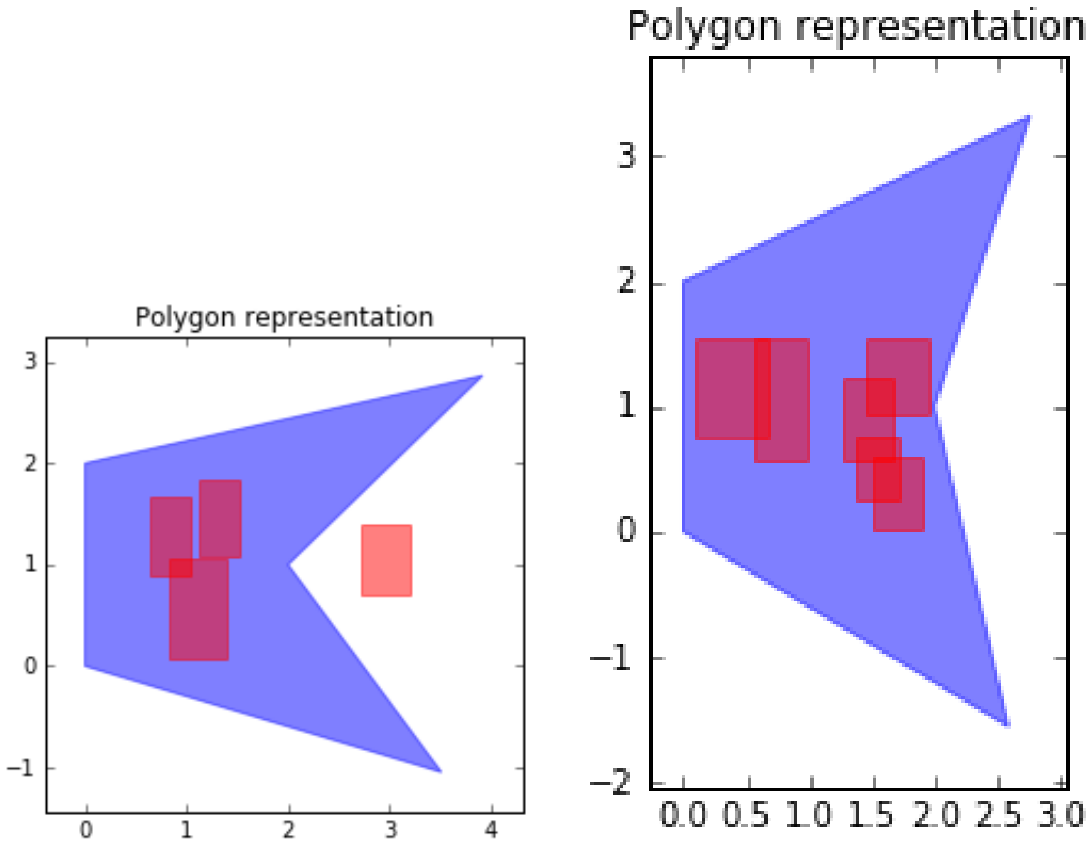


Figure 4.15: Two example samples of the third layout

separately.
training model 0: Layout 0 + target distance
training model 1: Layout 0 + minimum polygon overlay
training model 2: Layout 1 + closest side alignment
training model 3: Layout 2 + minimum centroid difference
training model 4: Layout 2 + surface ratio target
This naming convention also applies to the titles for the figures used to visualise the GSA. The format is ¡value for tested parameter¿, ¡training model¿, ¡name of the marginalised variable¿,¡sibling order of the GMM¿

Expressiveness preliminary discussion Before discussing the results I will start with a remark on the expressiveness in general. It is necessary in order to fully understand implications in later visualisations of it. For distributions with higher dimension, distributions for the training of siblings with higher orders (> 2) the expressiveness decreases tremendously. Below are examples from an evaluation which will be discussed later. Figure 4.2.3 visualise the bounds set by the user (red) and the marginalised distribution after training (green). From the figure it is straightforward to notice that the distribution surface coverage of sibling order 0 is much bigger than sibling order 4. This can partially be explained due to a lack of data for high fitness samples for higher sibling orders. As aforementioned it becomes increasingly hard to generate fit layouts when the number of objects increases. Because I use a simple rejection sampler the portion of high fitness samples is too low to train the GMM properly. In tests with significant difference in GSA over different parameters, I will focus the discussion on how the parameters can influence it. Figure 4.2.3 below visualises a trend which could be noticed across almost all tests. For the tests where it was not the case, I will mention it explicitly. The trend is a dramatic decrease in GSA for GMMs with sibling order larger than or equal to 3, when compared to GMMs with sibling 3 and smaller. In addition I would like to note that in general to compare

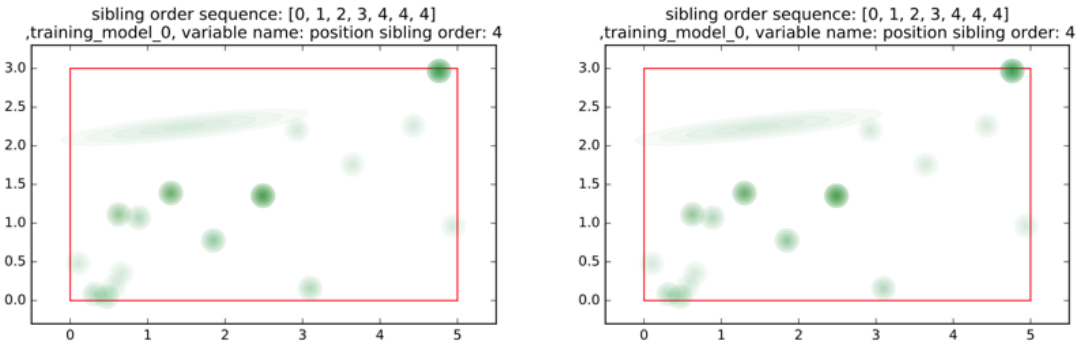


Figure 4.16: Comparison of distribution between sibling order 4 (left) and 0 (right)

expressiveness I will use the position variable. Because it is a variable which varies, and therefore can be visualised, for every layout used in the experiments. The density of the variables size and rotation are visualised as well and included in test files. However, I find them less informative. In figure 4.2.3 I give an example of the density for rotation (left) and size (right).

Rotation normalisation The rotation was scaled to 0-1 in the training data instead of 0-360. This was done because normalising data is standard practice in machine learning and the non-normalised rotation resulted in undesired expressive range visualised in figure 4.2.3 (left). The distribution contains a lot of very sharp local maxima and does not properly cover the user-defined range. In comparison can be seen that after normalisation the expressiveness improved (right).

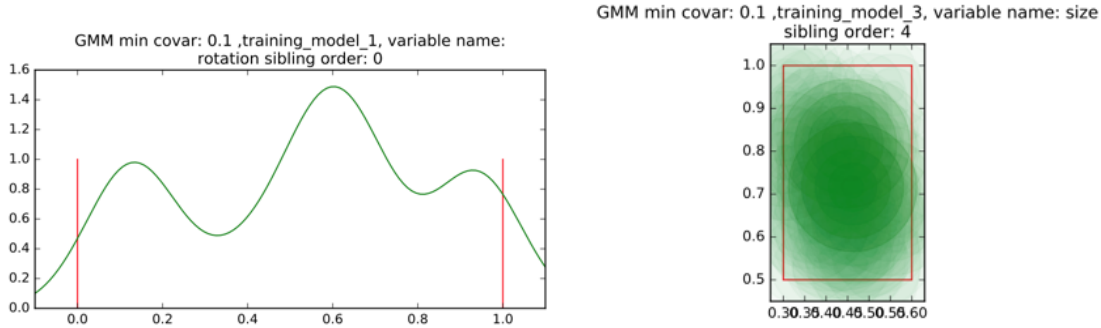


Figure 4.17: Visualisation of distribution of rotation and size

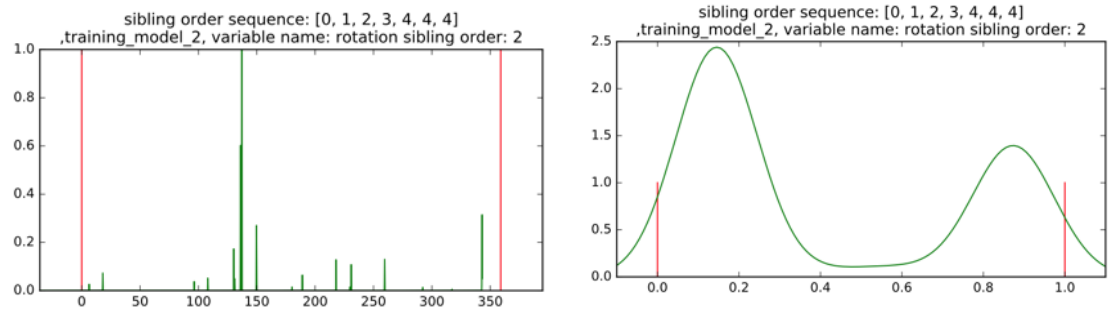


Figure 4.18: Comparison of rotation distribution without (left) and with (right) normalisation

Retraining parameters

In this section, I will evaluate the influence of the parameters that control the retraining process. The first parameter is the number of trials when training a new model, for every trial new data is generated. The second parameter is the number of retraining iterations. After obtaining a trained model it is possible to re-sample it and train again on this new data. The final parameter, and the reason why this test needs to be discussed first, is the number of samples when evaluating a newly trained model between trials and iterations. As mentioned in the paragraph 4.2.3 on the metrics and in the section on the retraining component int the methodology there can be variance on performance metric respectively due to the stochastic nature of EM and sampling bias. This test will give me an estimate on the variance, in order to identify the significance of the difference between two performance measurements.

Trials and iterations I start the evaluation of experiments always with the expressiveness in terms of GSA, followed by evaluating the performance and computation time.

Expressiveness In terms of expressiveness I highlight 3 results in figure 4.19. The first result (top left) is a distribution from a single iteration and single trial as baseline. In the second result (top right), the same variable and sibling order as the baseline are visualised. There is no significant change in GSA due to increase of the number of trials. The third result (bottom) is where the model was trained in 3 iterations and a single trial in each iteration. Here the overfitting, which results from training on data from a previously trained model, becomes apparent as the GSA significantly reduces.

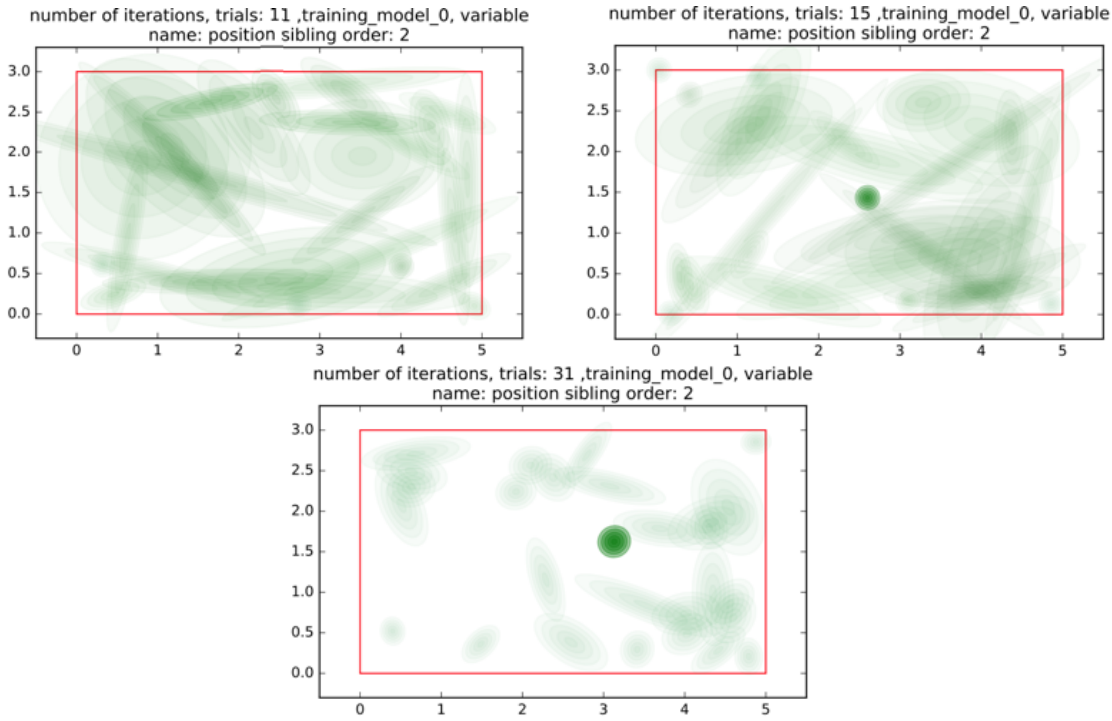


Figure 4.19: Expressiveness results for the experiment on trials and iterations

Performance and computation time No significant change in expressiveness was detected, as was to be expected. Because there was no change in parameters of the training method itself.

In this test computation is not included because the influence due to number of trials (n_{trial}) and iterations (n_{it}) can simply be given by $O(n_{trial}l * n_{it})$. The computation time of initialisation and finalisation is negligible. In the table 4.2.3 I show that the average fitness gain (AFG) due to increasing trials from 1 to 5 was 0.04, 0.05 and 0.03 respectively for 1, 2 and 3 iterations. This is barely above the significance threshold. In addition the AFG was not consistent over increasing trials, for example in 1 iteration the performance actually decreased from 4 to 5 trials. Combined increase of the trials and iterations results in some increase, but again barely above the significance threshold of 0.05 AFG, elaborated upon in the next paragraph. Therefore, I conclude that the number of trials has no significant influence on the performance.

Number of samples for evaluation between trials No significant change in expressiveness was detected, as was to be expected. Because there was no change in parameters of the training method itself.

trials		1	2	3	4	5	Average
	1	0.15	0.17	0.15	0.19	0.14	0.16
	2	0.14	0.19	0.18	0.17	0.18	0.172
	3	0.18	0.2	0.18	0.19	0.21	0.192
	Average	0.157	0.187	0.17	0.183	0.177	

Table 4.3: Performance and computation time results for trials and iterations experiment

In addition to the metrics on performance and computation time I measured the variance and standard deviation of AFG between 5 trials. As aforementioned, this standard deviation between trials will be used as the performance significance threshold. From the table 4.2.3 becomes apparent that the standard deviation slightly decreases with increasing number of evaluation samples, although not significantly. The standard deviations vary between 0.02 and 0.06, from which I conclude that any difference in AFG, the performance, above 0.05 can be considered significant and not due to uncontrollable randomness in the metric itself. I additionally conclude that the neither performance or computation time are significantly influenced due to changes in the number of evaluation samples.

Data generation parameters

In this section I will discuss the results for the parameters which influence the format or the amount of data used for the training the model. The first is the number of data samples generated in the training data for the training algorithm. The second test will evaluate training with and without Poisson disk sampling. Thirdly, I will discuss the the influence of the fitness function dimensionality when using probabilistic regression as learning technique.

Number of data samples The number of samples generated for the training data are a parameter which should intuitively lead to better results when increased, because, to my knowledge, more data is always better in machine learning.

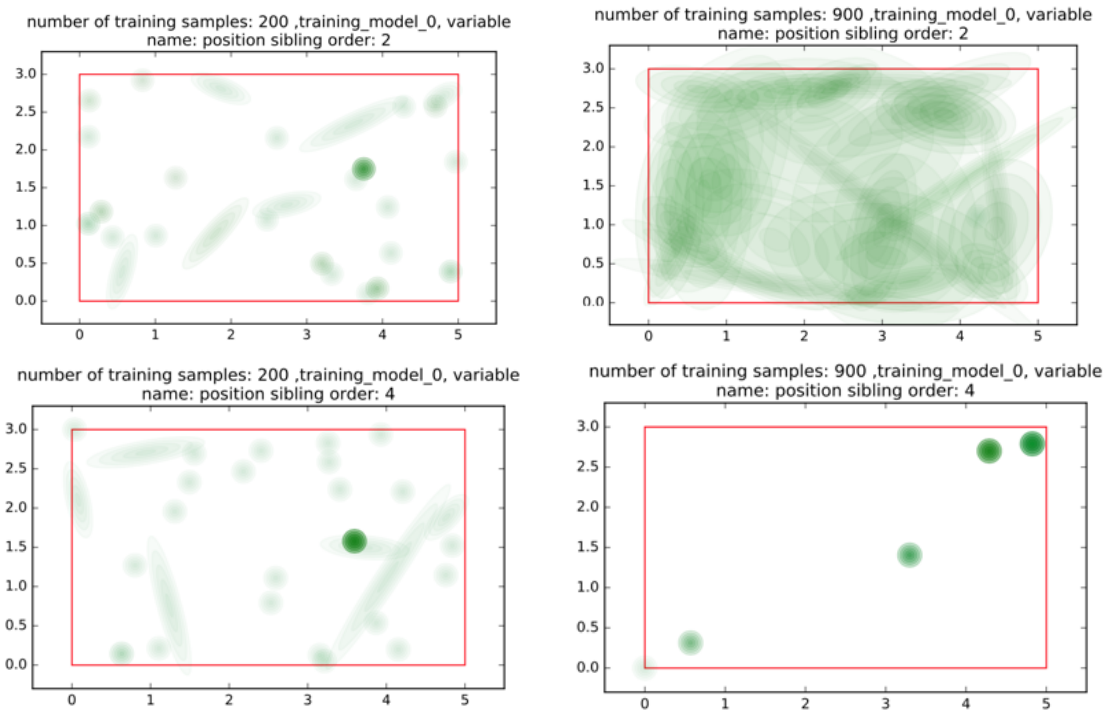


Figure 4.20: Expressiveness results for the experiment on number of training samples

Expressiveness From figure 4.2.3 I can conclude that increasing the number of data samples for training does improve expressiveness significantly. However if we compare expressiveness of variables from a higher sibling order, it becomes apparent that the improvement no longer

	fitness variance between trials	fitness standard deviation	average fitness gain	average computation time (s)
Number of evaluation samples				
200	0.00318	0.0564	0.16	225
400	0.00295	0.0543	0.19	220
600	0.000398	0.02	0.19	210
800	0.00147	0.0384	0.16	220

Table 4.4: Performance and computation time results for number training samples experiment

applies. This can be explained by the difficulty of generating samples with high fitness for higher sibling orders, as explained in the methodology. Thus even when increasing the number of generated samples the portion of "fit" samples for high sibling orders stays insufficient to properly train upon. This indicates that naive rejection sampling fails to generate high fitness samples for layouts with many instances. In order to better scale the methodology with increasing instances an optimised sampler like MCMC is desired.

Performance and computation time In table 4.2.3 can be seen that the AFG is maximum 0.06, between 300 and 900 data samples. This is barely above the significance threshold. Additionally, the AFG between 400 and 800 samples is only 0.01. Increasing the number of data samples does however increase stability of the score. This is because the standard deviation, due to sampling bias, between consecutively trained models over different trials has decreased. The results per model are available in the experiment files. The computation time however does increase with the number of data samples. But the increase is relatively slow and not consistent. I conclude, that even though the performance does not increase significantly, it is advisable to set the number of training samples as high as possible because it increases stability at a small computational cost.

Poisson disk sampling Poisson disk sampling (PDS) was proposed to have a more uniform samples from the user-defined search space. This could reduce the aforementioned sampling bias.

Expressiveness I will first evaluate the expressiveness by comparing the GSA with and without PDS.

I initially expected PDS to improve expressiveness because more uniform samples would result in a better coverage of the search space. However, this was not reflected consistently in the visualisation of GSA. For example in figure 4.2.3, the top two visualisations show a case where some improvement can be seen with PDS(left) , whereas the bottom two visualisation shows the exact opposite.

Performance and computation time In the table 4.2.3 I give an overview of the performance and computation time for the experiment on PDS.

Number of data samples	average fitness gain	average computation time (s)	Computation time difference with previous result
200	0.15	210	/
300	0.14	235	25
400	0.17	249	14
500	0.15	253	4
600	0.18	270	17
700	0.19	282	12
800	0.18	279	-3

Table 4.5: Performance and computation time results for number of samples for evaluation between trials experiment

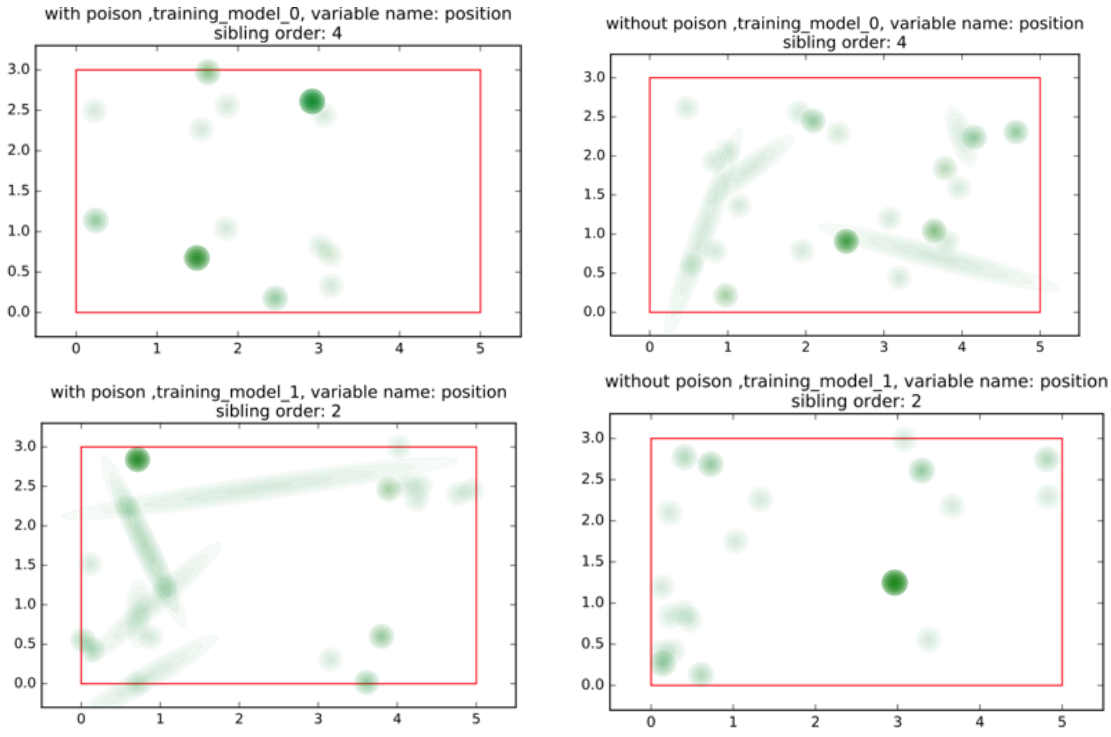


Figure 4.21: Expressiveness results for the experiment on Poisson disk sampling

	average fitness gain	average computation time (s)
With PDS	0.16	619
Without PDS	0.15	213

Table 4.6: Performance and computation time results for Poisson disk sampling experiment

PDS does not influence the AFG significantly. It does however drastically increase the computation time. I conclude that the use of PDS in my methodology does not offer any apparent advantage.

Fitness dimensionality reduction in probabilistic regression As mentioned in the methodology, the learning algorithm probabilistic regression can use multi-dimensional labels for the supervised density estimation. I will evaluate whether any if the proposed dimensionality reduction methods; single fitness dimension, sample fitness dimension, full fitness dimension, can influence the metrics.

Expressiveness I start by comparing the GSA between the different methods given in figure 4.2.3.

In the first three results of figure 4.2.3 I show that using regression drastically improves the GSA in terms for higher sibling order distributions. There was however no difference between the different methods of dimensionality reduction for probabilistic regression. The last two results show that even for lower sibling orders the expressiveness improves. I conclude that the use of probabilistic regression is necessary as learning algorithm for the hyper-generator, in order to offer maximum expressive range. However, weighted density estimation (WDE) is the default learning algorithm for the experiments. Therefore, even though I will still evaluate GSA changes for each experiment, the conclusions concerning expressive range over different parameter values should be interpreted as information on the behaviour of the parameter and not the methodology as a whole. The larger expressive range for higher sibling orders in probabilistic regression can be explained by formulating it as fitting a GMM curve through the data of both the asset samples and their fitness. While WDE will only fit the GMM to data of the asset samples. In a part of the search space where less data is available, as is the case for harder to generate high fitness samples, "GMM curve fitting" can estimate the values of missing points by interpolating. Whereas the WDE will treat the lack of higher fitness data samples as a less dense region and thus reflect it in the trained model, i.e. smaller GSA.

Performance and computation time Even though the expressiveness improved with probabilistic regression, it is still important to deliver significant performance with it.

In table 4.2.3 I show that in terms of AFG the weighted density estimation performed slightly better than probabilistic regression, although not significantly. The computation time is not consistently different between methods. I conclude that probabilistic regression has similar performance as weighted density estimation and should therefore be favoured over weighted density estimation, due to its better expressiveness.

Data filtering with fitness function threshold In order to increase the number of high fitness samples in the training data I added a possibility to filter out samples which had a fitness value below a certain threshold.

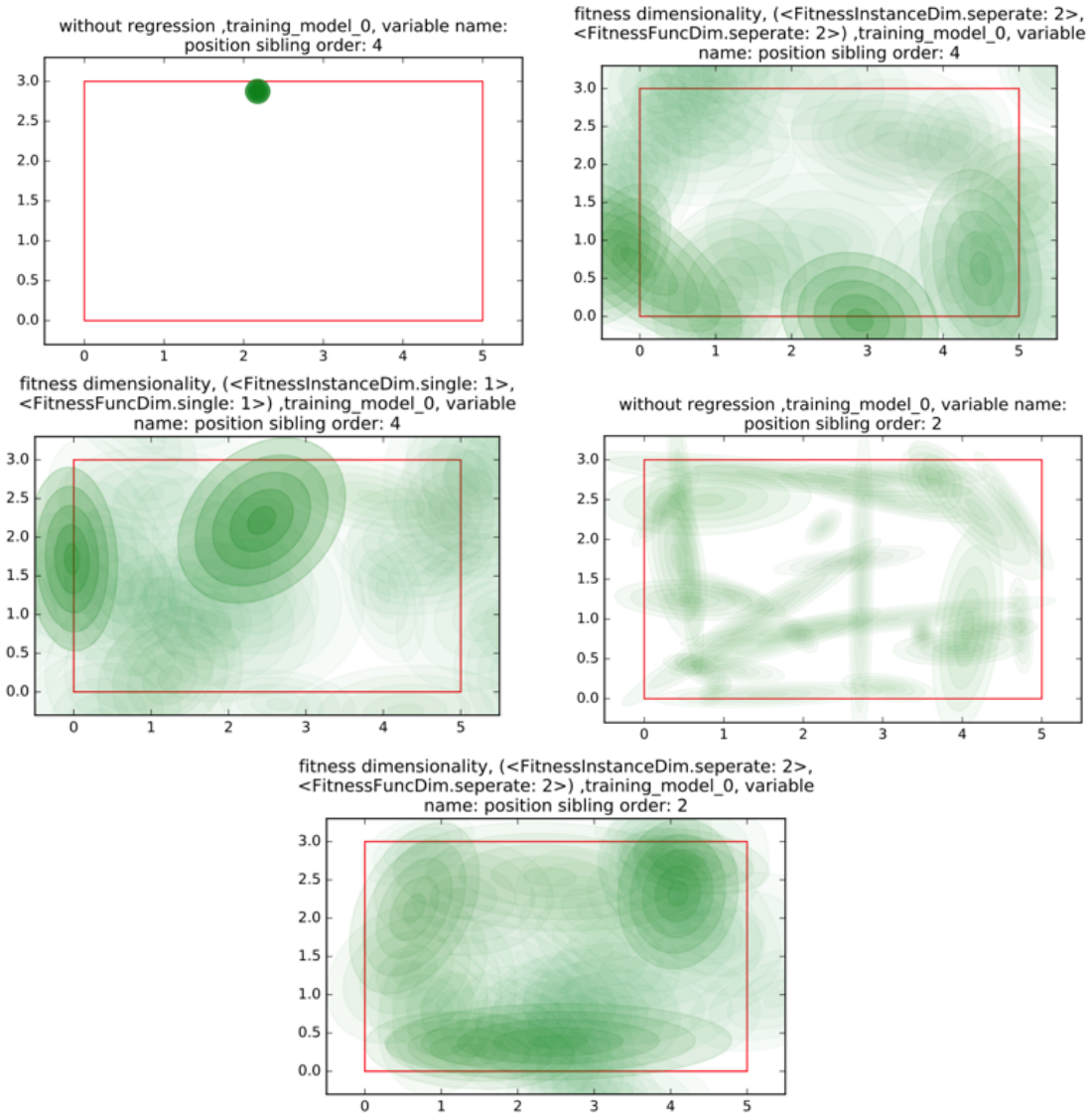


Figure 4.22: Expressiveness results for the experiment on fitness dimensionality reduction

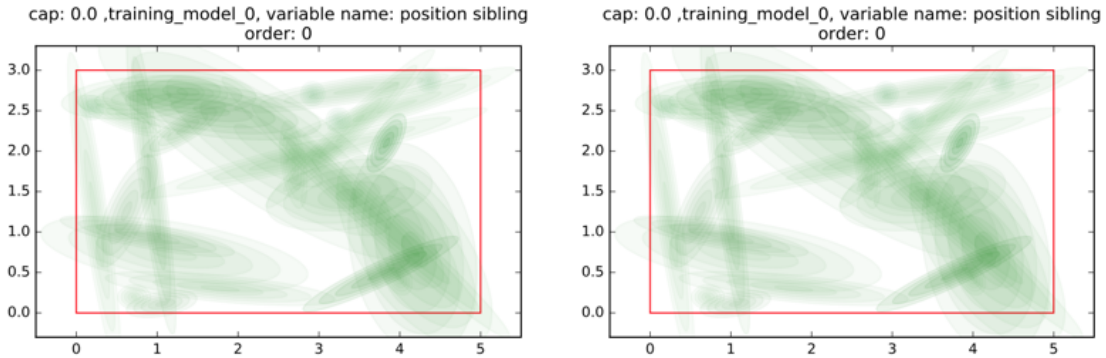


Figure 4.23: Expressiveness results for the experiment on data filtering

Dimensionality reduction, with probabilistic regression	average fitness gain	average computation time, seconds
full fitness dimension	0.12	246
sample fitness dimension	0.12	214
single fitness dimension	0.12	233
Weighted density estimation		
single fitness dimension	0.14	222

Table 4.7: Performance and computation time results for fitness dimensionality reduction experiment

Expressiveness There was no significant change in expressiveness due to different fitness thresholds. Figure 4.2.3 give an example of distribution with different caps and very similar GSA.

Performance and computation time In table 4.2.3 can be seen that contrary to what might be expected, increasing the average fitness in the example samples used for training does not significantly increase the AFG. Neither can I deduce any influence in computation time.

Fitness parameters

Some parameters can be configured separately for each fitness function. The first is the fitness polynomial order applied to its values when training. I will evaluate this by comparing the influence of the order on an adapted version of training model 0, with two instead of only one fitness function. The second parameters is the fitness target, which is used for fitness function that don't need to be maximized but for which the difference with a certain target should be minimized, called targeting fitness functions. This will be evaluated for training model 4.This model has been chosen because its (targeting) fitness function was the hardest to optimise and for which I suspected the cause to be its fitness target. Thirdly, I evaluate whether changing the hierarchical relation from pairwise siblings to pairwise parent-child influences the results for training model 2. This training model has been chosen because it had a fitness function that did not optimise well and which supported both types of fitness hierarchical relations.

Fitness polynomial order For the evaluation of the polynomial order I added the fitness function minimum polygon overlay to training model 0. In summary the experiment will evaluate training model 0 with fitness function; target distance and minimum polygon overlay, respectively TD and PO in the title of the figures. With fitness polynomial order varying separately for each function, over the range $[4, 28]$ with step size 4. This range has been decided upon after experimentation, in order to have significantly varying results in the metrics.

Expressiveness To evaluate the GSA, I first compare the top left visualisation in figure 4.2.3 with the figure4.2.3 from previous experiment, which both visualise training model 0 4.2.3. The difference being that in figure 4.2.3 the train model has two fitness functions instead of one. I show that the GSA is significantly reduced. In addition I show that changing the respective

Fitness threshold	Average fitness gain	Average computation time, seconds
0	0.1	157
0.05	0.16	192
0.1	0.17	180
0.15	0.14	191
0.2	0.15	190
0.25	0.14	227
0.3	0.16	195
0.35	0.15	206

Table 4.8: Performance and computation time results for the data filtering experiment

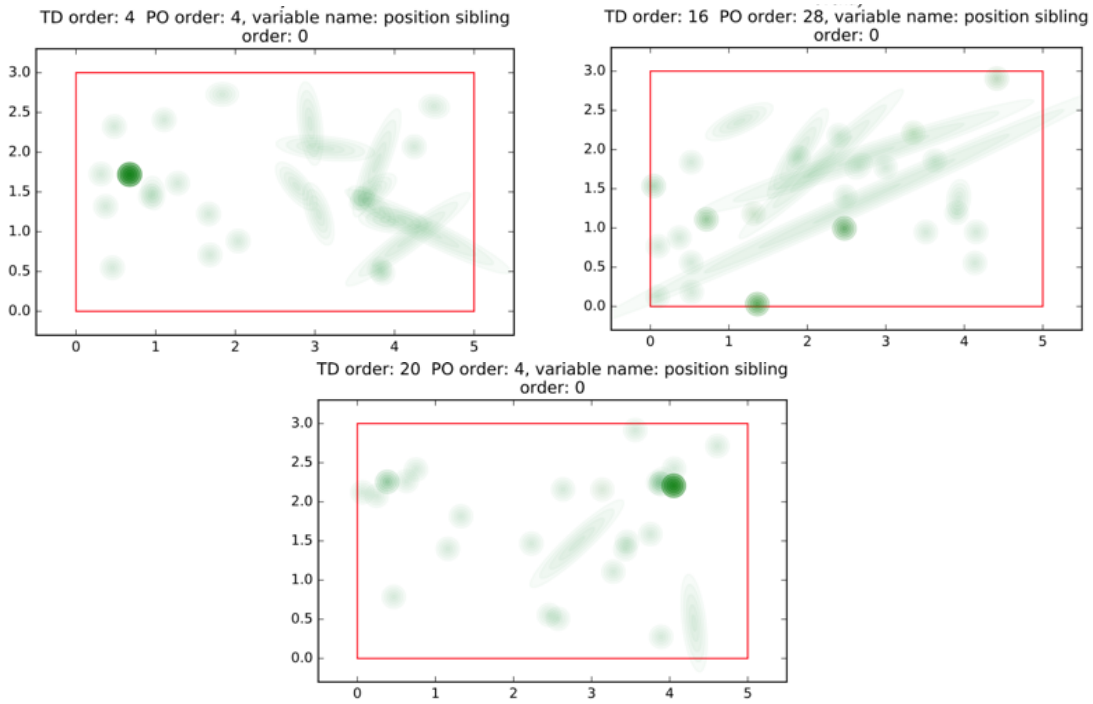


Figure 4.24: Expressiveness results for the experiment fitness polynomial order

order of the fitness functions does not significantly influence the GSA. As can be seen on the results in figure 4.2.3 (top right and bottom). I conclude that training on two fitness functions simultaneously does have a negative influence on the expressiveness.

Performance and computation time For this test I did not include computation time as it did not significantly change. In table 4.2.3 I show that in most order combinations the polygon overlap has been improved but the minimum distance fitness is not and even significantly below the baseline value (0.6 AFG) in some cases (red), this means that there was a loss instead of a gain in fitness average. In other cases there was no significant improvement at all (green). And there is not a single case where the PO has improved more than the TD. I conclude that even though it is possible to optimise fitness functions simultaneously, by carefully picking its order, it is possible for a fitness to constantly dominate the other. Additionally, I am not able to find a clear pattern for the order combinations which consistently results in higher performance. This conclusion is an additional motivation to evaluate the fitness gain for each fitness function in a separate layout.

Fitness target As aforementioned the fitness target is a parameter which can be configured for each fitness function. This user-defined parameter changes which value of the fitness function will result in higher fitness. For example, the user can set the target to distance between siblings to either 1 or 3. However it does not directly influence the training process, it only changes the behaviour of the data extracted from the fitness function. This is a more ad-hoc experiment, motivated by the lack of optimisation in training model 4 4.2.3 with fitness function surface target ratio. For this fitness function I did not achieve any significant performance gain. Over all parameter experiments combined, the performance never exceeded the minimum significance threshold of 0.05 . I suspected that optimisation was not possible due to an incompatible configuration of layout and fitness function, which could be resolved with a different target for the fitness function.

There was no significant change in expressiveness and figures are omitted for brevity.

Performance and computation time In table 4.2.3 I show that the maximum AFG is 0.05. I conclude that even varying the target values does not result in significant performance gain for training model 4.

Fitness hierarchical relation This experiment will also be an ad-hoc evaluation of a single fitness function. In this experiment I evaluate whether changing the hierarchical relation for the fitness function closest side alignment 4.2.3 could influence the performance.

Performance and computation time In table 4.2.3 I show that there is barely any difference in AVG. I conclude that changing the hierarchical relation does not help in optimising the fitness function closest side alignment.

Hierarchical model parameters

There are two parameters which influence the hierarchical structure of the model, both used to reduce computation time. First is the sibling order sequence which defines for each child how high its order of conditional dependency on previous siblings is. The second is the amount of marginalisation required in order to have a GMM for each sibling order. This parameter is a sequence of booleans where True means that a separate GMM is trained for that order in the

	Polygon overlay (PO) order							
target distance (TD) order		4	8	12	16	20	24	28
	4	0.59,0.58	0.6,0.88	0.67,0.56	0.60,0.49	0.56,0.68	0.5,0.69	0.57,0.59
	8	0.69,0.58	0.54,0.6	0.57,0.63	0.6,0.61	0.6,0.45	0.53,0.65	0.64,0.65
	12	0.59,0.65	0.6,0.42	0.61,0.67	0.6,0.42	0.59,0.50	0.59,0.6	0.63,0.42
	16	0.61,0.8	0.54,0.56	0.56,0.62	0.61,0.74	0.61,0.6	0.59,0.67	0.52,0.77
	20	0.6,0.63	0.58,0.57	0.57,0.62	0.62,0.64	0.52,0.62	0.63,0.61	0.5,0.78
	24	0.52,0.63	0.64,0.76	0.57,0.67	0.61,0.58	0.57,0.66	0.5,0.57	0.6,0.69
	28	0.48,0.63	0.57,0.61	0.58,0.65	0.55,0.61	0.67,0.49	0.54,0.65	0.56,0.51

Table 4.9: Performance and computation time results for the experiment on fitness polynomial order

model 4 target (for fitness surface target ratio)	Average fitness gain
0.05	0.017
0.15	0.03
0.25	0.04
0.35	0.05
0.45	0.04
0.55	0.04
0.65	0.03
0.75	0.04
0.85	0.04
0.95	0.0074

Table 4.10: Performance and computation time results for the experiment on fitness targets

fitness relation	
parent child	Average fitness gain
	0.007
sibling	
	0.009

Table 4.11: Performance and computation time results for the experiment on fitness hierarchical relations

sequence, and False means that the GMM is marginalised from a GMM of the first higher sibling order.

Maximum sibling order The variable sibling order is evaluated by increasing the maximum sibling order with steps 1 of and using a sequence where the each sibling depends on all its previous sibling as long as it does not exceed the set maximum.

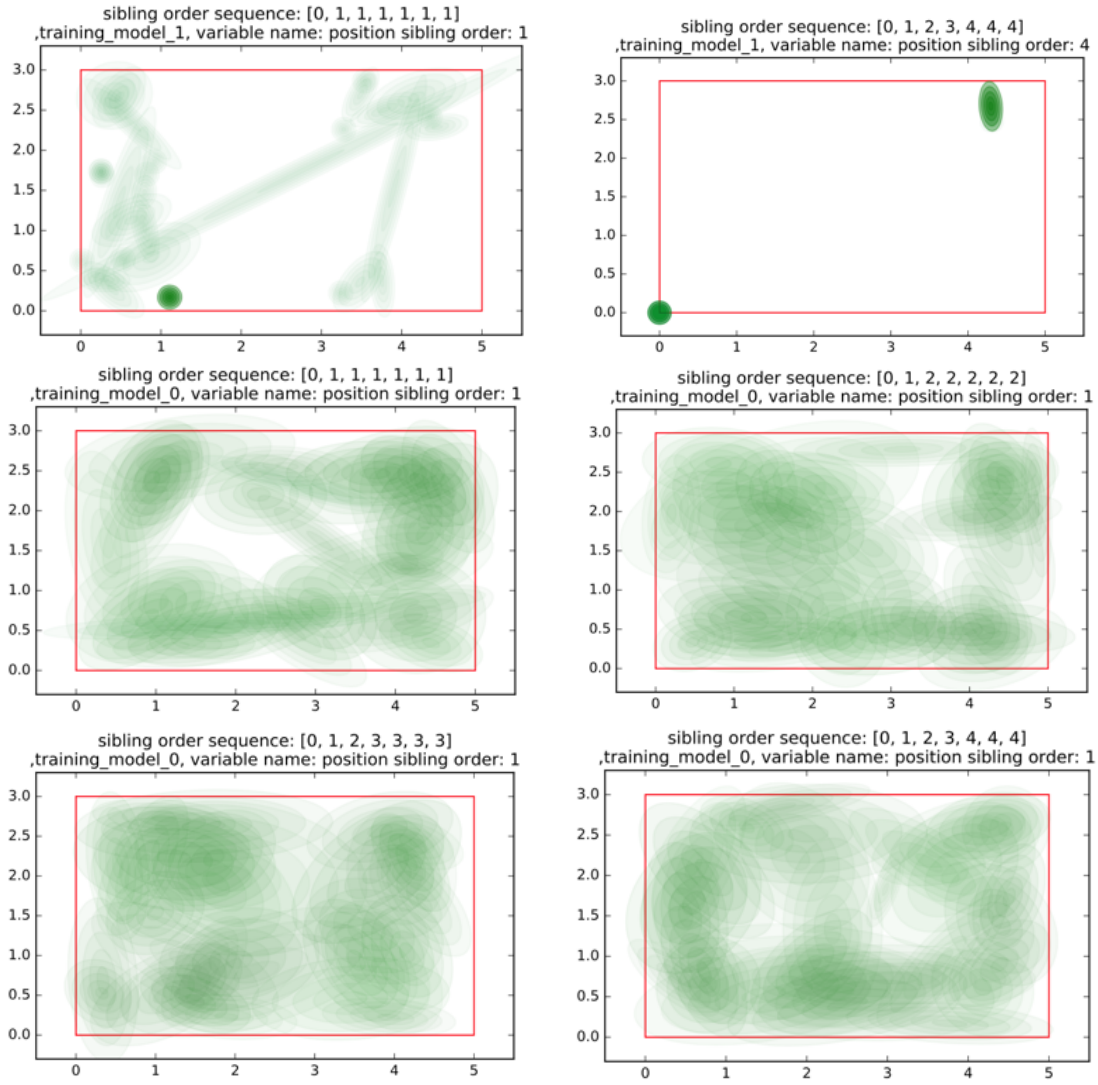


Figure 4.25: Expressiveness results for the experiment on maximum sibling order

Expressiveness In figure 4.2.3 there is no significant decrease in GSA for increasing sibling order between the examples. I conclude that there is no significant difference in expressiveness by increasing the maximum sibling order. However, there is difference in form of the GSA between different sequences. This can be explained due the learning algorithm, which can have comparable results with similar GSA but different parameters for its respective GMMs.

Performance and computation time To ensure that the marginalisation of GMMs does not interfere with the performance metric, I trained a separate GMM for every sibling order. In table

sibling order sequence	average fitness gain	model average computation time (s)
[0,1,1,1,1,1,1]	0.17	160
[0,1,2,2,2,2,2]	0.18	212
[0,1,2,3,3,3,3]	0.18	239
[0,1,2,3,4,4,4]	0.16	273

Table 4.12: Performance and computation time results for the experiment on maximum sibling order

4.2.3 can be seen that increasing the number of siblings did not increase the AFG. Intuitively, by training more GMMs the computation time increases as well. Therefore I conclude that reducing the sibling order will reduce computation time without hurting the performance or the expressive range. Specifically, the computation time can be reduced on average with 10% for each reduction in sibling order. (i.e., $0.9 \cdot O(n) = O(n-1)$)

GMM marginalisation For the gmm marginalisation I will change which sibling order for the hierarchical model will have a separately trained GMM. The last sibling order is kept to True. The highest sibling order cannot be marginalised from a higher order because there is none.

Expressiveness I conclude that marginalisation has a significant influence on expressiveness. As example I give two extremes in figure 4.2.3 where the maximum amount of separately trained GMMs (row 1 and 2 left) is compared to the minimum (row 1 and 2 right). Additionally, in figure Y is shown that maximum marginalisation can even lead to a GSA of almost 0. The last 3 results show the influence more in detail. If a gmm is marginalised from a higher order GMM (row 3 left) which itself has low expressive range (row 3 right) than its expressive range will be low as well compared to when it is trained separately (row 4).

Performance and computation time In table 4.2.3 I show a somewhat surprising result is that even though marginalisation sometimes reduces the expressive range to 0, on average it still results in acceptable performance. Overall, the change in performance by changing the marginalisation of GMMs is barely significant. There is however no significant difference in computation time.

GMM parameters

The parameters which are evaluated of the underlying probabilistic model are its number of mixture components and the minimum covariance. In literature on machine learning, the process of finding the optimal model parameters is called model selection. Model selection is often done using cross-validation and an information criterion, for example the Bayesian information criterion (BIC). Even though this approach could be applied to choose the parameters in my methodology, for evaluation I prefer to discuss the influence of these parameters in the same

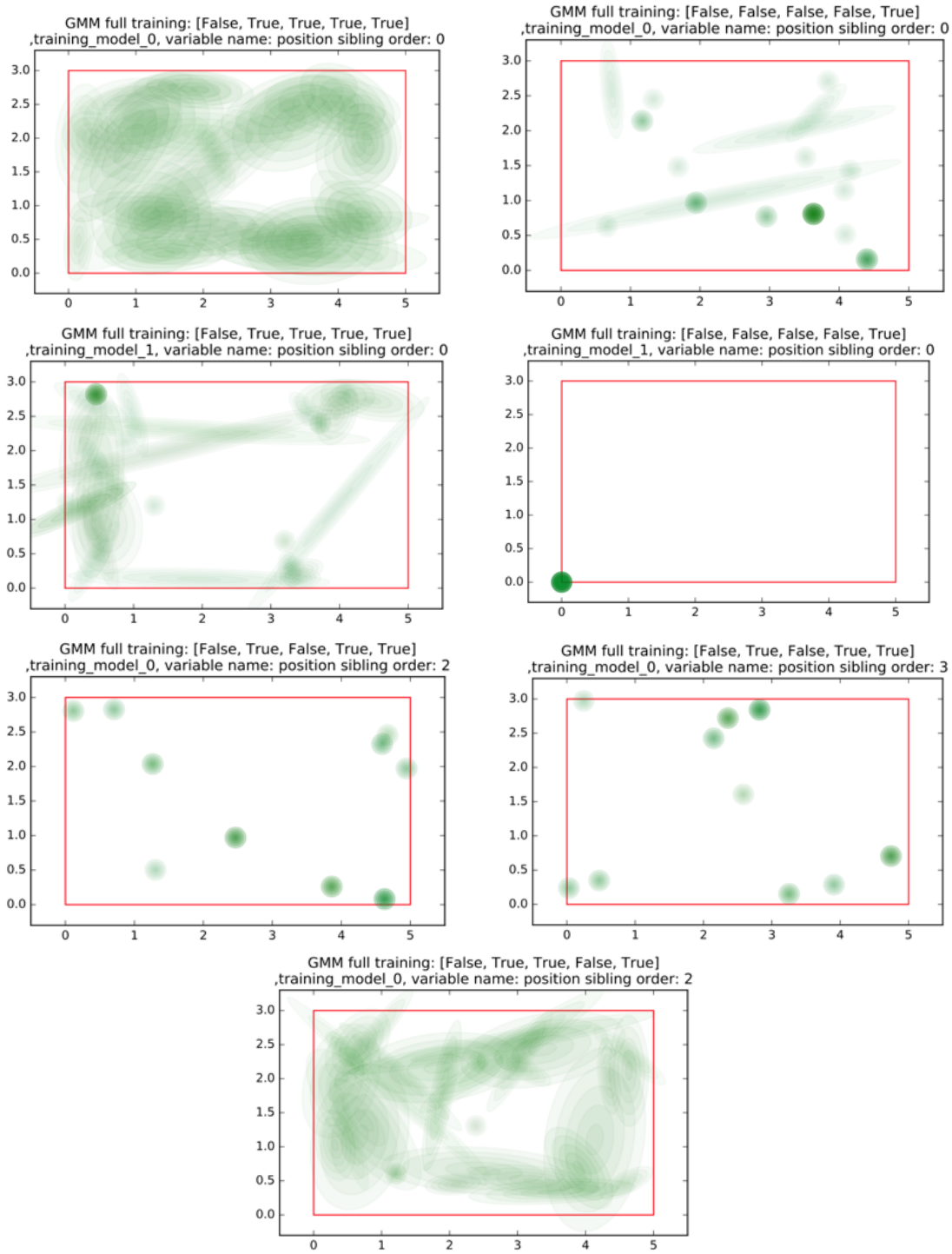


Figure 4.26: Expressiveness results for the experiment on marginalisation

full gmm training per sibling order	Average fitness gain	Average computation time (s)
[False, True, True, True, True]	0.16	264
[False, True, True, False, True]	0.17	238
[False, True, False, True, True]	0.16	243
[False, True, False, False, True]	0.15	218
[False, False, True, True, True]	0.19	245
[False, False, True, False, True]	0.18	226
[False, False, False, True, True]	0.13	232
[False, False, False, False, True]	0.17	245

Table 4.13: Performance and computation time results for the experiment on marginalisation

metrics as used for the other parameters in my methodology. This applies to the first and second parameter, respectively number of mixture components and minimum covariance. The third parameter is of my own design and only applies to training with probabilistic regression. It is the condition value for the fitness function vector (fitness data), for which the density has been estimated jointly with the sample data. As mentioned in the methodology section on probabilistic regression, this joint density needs to be conditioned on certain fitness values in order to derive the separate density of the samples.

Number mixture components Increasing the amount of components in the mixture, increases the number of parameters of the probabilistic model thus also the possibility of overfitting. Overfitting is not desired, but because the methodology is not a standard learning problem, the precise implications of overfitting are not immediately clear. Overfitting is often measured with cross-validation and separate training and test data sets. However, my setup is different in the sense that the there is no unseen data, I can generate as much data as I want, at a computational cost. Thus, using cross-validation would not make much sense for any of my tests. Additionally, the environment is noiseless in a sense that all labels of the samples, the fitness function values, are exact. I expect overfitting is to increase the performance and decrease the expressive range.

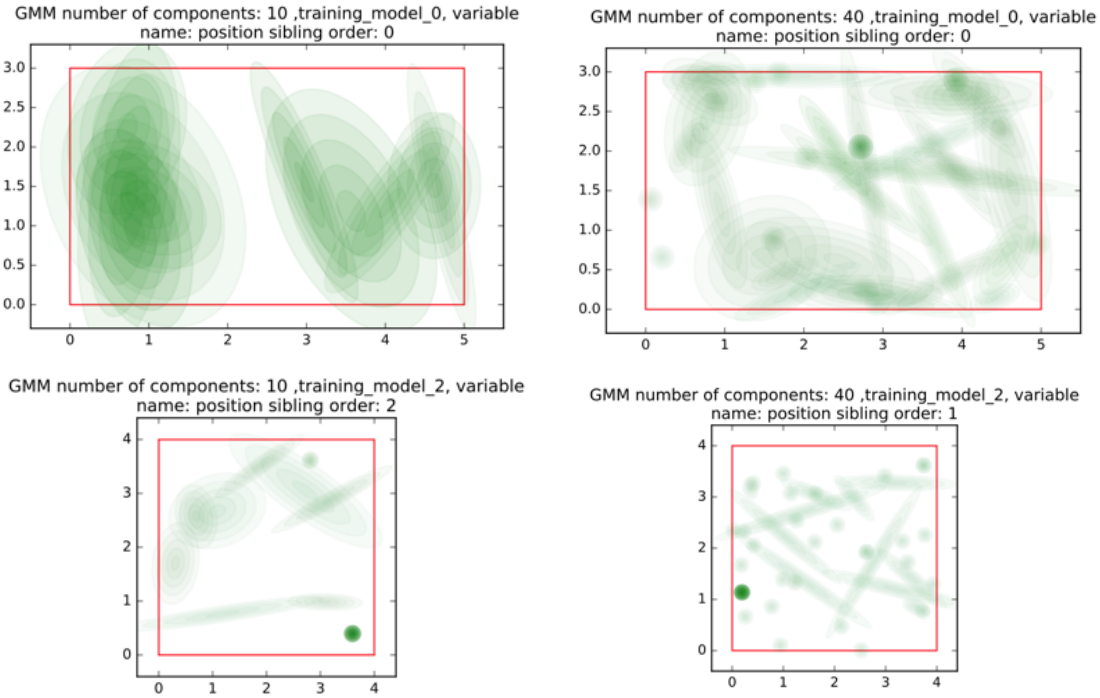


Figure 4.27: Expressiveness results for the experiment on number of mixture components

Expressiveness When comparing the visualisations in figure 4.2.3 I show that more components result in different forms of GSA. More specifically increasing components, results in a GSA form with a seemingly higher resolution estimate of the fitness function. But because I do not know the exact behaviour of the fitness function, it is very hard to deduce whether the increase of resolution is a desired trend.

Performance and computation time As shown in table 4.2.3 increasing the number of parameters in the GMM, increases the computation time. The AFG however, does not increase

Number of components	Average fitness gain	Average computation time, (s)
10	0.16	129
20	0.2	171
30	0.15	227
40	0.16	269

Table 4.14: Performance and computation time results for the experiment on number of mixture components

significantly. The additional components did not help in optimising the the fitness function. Even though the distribution with more components seem more precise (figure 4.2.3), I conclude that the added resolution can be regarded as noise because increasing the components did not result in an increased performance.

Minimum covariance Even though the minimum covariance does not change the number of parameters in the GMM. It is a parameter which prevents overfitting by defining when a factor in the covariance matrix found during training is the result of noise in the data [99].

Expressiveness As shown in figure 4.2.3, both for sibling orders (2) with reasonable GSA (left) and higher sibling orders (3) with less GSA (right). Therefore I conclude that changing the minimum covariance does not significantly influence the expressiveness.

Performance and computation time In table 4.2.3 I show that there is a slight performance gain when increasing the minimum covariance, however barely significant (0.06) and not consistently. The computation times stays approximately the same. I conclude that the minimum covariance can for reasonable values $[0.1 - 0.00001]$ does not influence the metrics.

Regression fitness condition This regression condition can be configured for each fitness function separately. The range of this parameter is $[0.6, 1.2]$, set after initial experimentation, because the actual fitness we want to achieve is 1 (the normalised maximum) and I want to evaluate whether changing the fitness conditioning value would have an influence.

Expressiveness In figure 4.2.3 I compare the difference in GSA between condition value 0.6 (left) and 1.2 (right) in training model 0 (top) and training model 3 (bottom). There is no significant difference in expressive range due to change in the fitness condition value. Even the form of the GSA for in figure X stay approximately the same.

Performance and computation time I can conclude from table 4.2.3 that there is no significant change in performance or computation time.

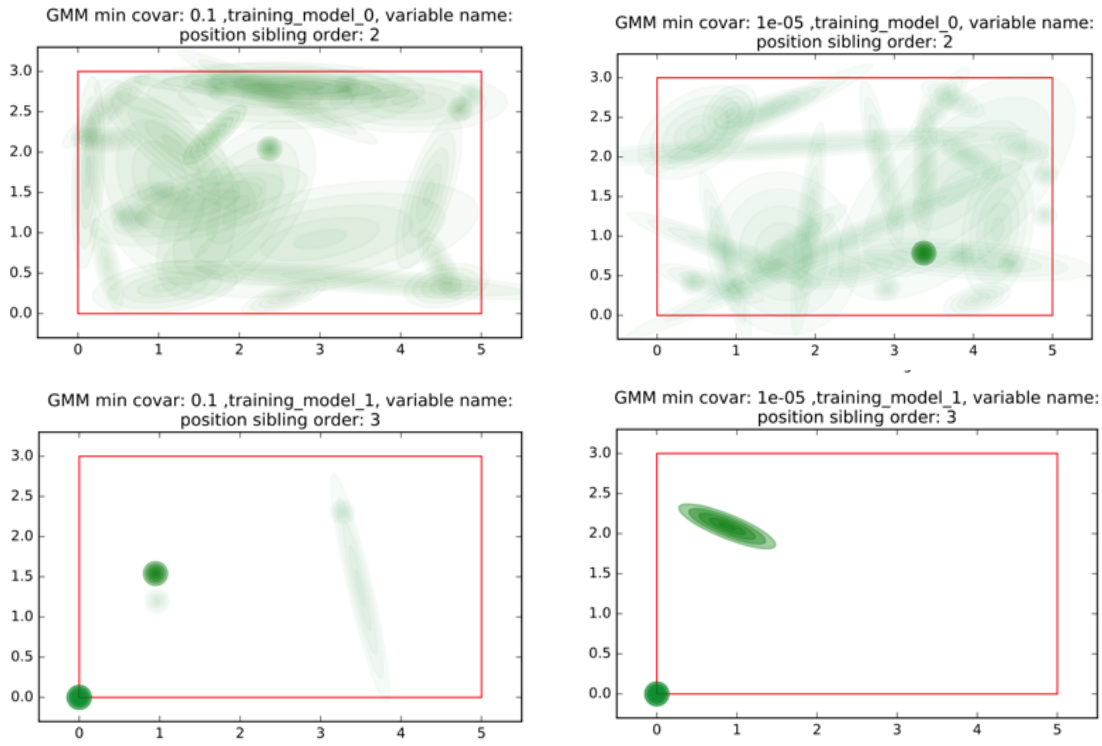


Figure 4.28: Expressiveness results for the experiment on minimum covariance

gmm min covar	Average fitness gain	Average computation time, seconds
0.1	0.18	143
0.01	0.2	139
0.001	0.16	139
0.0001	0.17	140
0.00001	0.16	140
0.000001	0.13	136

Table 4.15: Performance and computation time results for the experiment on minimum covariance

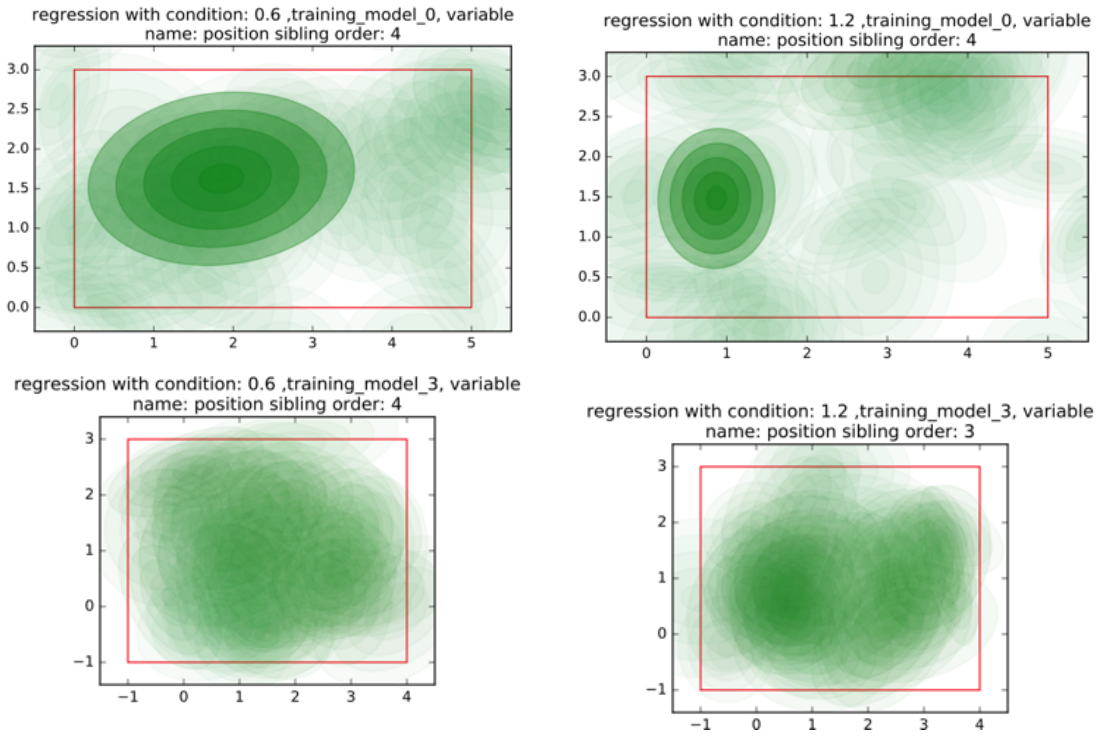


Figure 4.29: Expressiveness results for the experiment on regression fitness condition

regression condition	average fitness gain	model average computation time, seconds
0.6	0.12	244
0.7	0.13	244
0.8	0.12	219
0.9	0.12	219
1	0.12	221
1.1	0.11	246
1.2	0.12	247

Table 4.16: Performance and computation time results for the experiment on regression fitness condition

Training model configuration

As mentioned in paragraph 4.2.3 every training model has two stochastic variables for its children. I evaluate the influence of removing one of the variables from the training data, and therefore excluding the information it could yield from the learning process. The excluded variables is always that one which is not directly used as input for the influence. For example, in training model 2 with the fitness function closest side alignment is estimated, the position variables is omitted and the rotation variable is kept. The position indirectly influences the fitness because, it is based on the position that the closest side between two polygons is decided. The motivation behind this experiment was to evaluate whether the influence of a variable which indirectly influences the fitness function could be learned as well.

Number of training variables Adding variables leads to a higher dimensionality in the probabilistic model. This is also the case when the sibling order is increased. Because, as aforementioned in the methodology, the dimensionality of the GMM increases linearly with the number of sibling orders and number of variables. Therefore, the influence of increasing dimensionality has already been discussed. In favour of brevity I will only compare between training with only a single child variable and a two child variables.

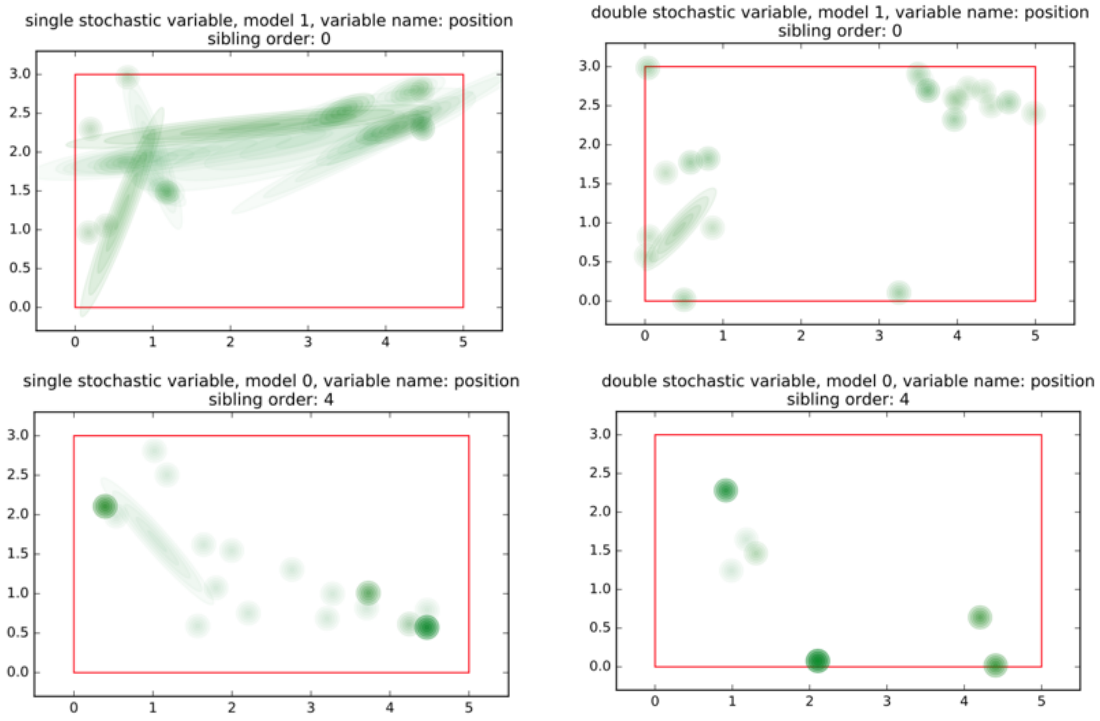


Figure 4.30: Expressiveness results for the experiment on number of training variables

Expressiveness In figure 4.2.3 I compare the GSA for training model 1 (top) and training model 0 (bottom), with training on a single (left) and with double number of variables (right). The GSA slightly decreased for the models with two variables. This can be explained because (1) both probabilistic models are trained on the same amount of training data, while the model with two variables has more parameters to estimate for its GMM and (2) I have concluded in section 4.2.3 that adding training data increases expressiveness. Therefore I conclude that adding a variable to the training data should be combined with increasing the training data in order to keep the same expressive range

Performance and computation time In table 4.2.3 is shown that the average fitness did not significantly improve with the additional data from the second stochastic variable. Surprisingly, neither is the computation time. However, because the test is limited to only one or two variables I will not generalise the performance results to layouts with a larger amount of variables.

4.2.4 Learning of fitness functions

In summary I was able to significantly increase the average fitness value over all parameter configurations for the fitness functions target distance (22%), minimum polygon overlap (100%) and minimum centroid difference (45%). The other fitness functions; closest side alignment and target surface ratio, were harder to optimise and only reached respectively 5 and 10% improvement in the best of cases over all parameters. This was partially the motivation to try different, ad-hoc, experiments for these fitness function but without positive result, see paragraph 4.2.3 4.2.3.

single variable	model average score gain	model average computation time, seconds
	0.11	57
double variable		
	0.12	59

Table 4.17: Performance and computation time results for the experiment on training variables

Chapter 5

Conclusion and future work

5.1 Conclusion

5.1.1 Contributions

I have presented a new approach which is able to optimise the search space of a content generator for some fitness functions, which will be elaborated in following paragraph. This is achieved by using a learning algorithm which automatically adapts a user-defined search space, with a minimal decrease in expressive range of the generated content. The search space, defined by a probabilistic model, combined with the fitness functions offer the user two distinct types of control in order to increase controllability, respectively a bottom-up and top-down type of control. The automatic search space optimisation allows the user to define the search space in a coarse way, without the need to worry about the implications on computational performance.

I have introduced a new metric for the expressive range of content generators based on the probability density function of the random variables in the search space, more specifically the bias and variance of their respective densities.

The approach used 5 distinct fitness function from an architectural context for generating scene layouts, namely (1) minimum polygon overlap, (2) target surface ratio, (3) target distance, (4) minimum centroid difference, (5) closest side alignment. I have shown that is possible to significantly optimise the search space for fitness functions 1,3 and 4.

I have shown that increasing the number of samples in the training data for the learning algorithm, significantly increases performance and expressive range. While only increasing computation time with 50% when quadrupling the number of samples. Additionally I have shown that probabilistic regression significantly outperforms weighted density estimation as supervised density estimation in terms of expressive range of the content, with similar results in terms of performance and computation time.

Finally, I proposed an optimisation in my methodology to reduce the computational complexity by reducing the variable sibling order in the probabilistic model. This resulted in a 10% drop in computation time for each reduction in maximum sibling order (i.e., $O(max_{sibling_o rder-1}) = 0.9 * O(max_{sibling_o rder})$).

5.1.2 Limitations

The probabilistic model in my methodology is restricted to continuous random variables and not suitable for discrete random variables. Additionally, the probabilistic model is a probabilistic graphical model with acyclic and directed structure which cannot model cyclic generation process' like grammars or undirected generation process' like sampling from a factor graph.

The system was partly motivated as tool which integrated learning algorithms and still be accessible to non-technical users. However, some parameters of the learning process require complex evaluation and technical knowledge to be tuned. Additionally, one parameter, the fitness function polynomial order, also has significant but less predictable influence on the system's performance.

5.1.3 Perspective

I think that my methodology, the hyper generator, of combining a probabilistic model, high level constraints and a learning algorithm which adapts the first to the lather, is sound and would be a strong alternative to other existing approaches for PCG. However, the results from current implementation, the specific choice of probabilistic model and learning algorithm, are not significant enough to choose my method over standard search algorithms without adaptive search space, for example MCMC sampling. In order for the hyper generator to be a considered as alternative, I envision a more complex and suitable probabilistic model, which would be capable of capturing a wider range of fitness function with high accuracy. Therefore, I also suggest to drop any optimisations which reduce the model's complexity, in favour of reduced computation time.

5.2 Future Work

In the generation process the user needs to explicitly state the hierarchical relations between asset variables. These can be clear, for example in the case of chairs and tables, but it will not be for all types of content. A good solution would therefore be to learn the hierarchical relations from the data. This would (1) relieve the work of defining a hierarchy from the user and allow a user to apply a fitness function between two objects which are only implicitly related. For example a fitness on pieces of furniture within a certain proximity of each other (implicit), instead of a fitness between furniture A and furniture B (explicit).

Games can have more complex fitness function than those discussed in the context of architecture, for example the behaviour of an AI playing the game as fitness function [100]. These fitness functions can apply not only to static objects in a level but also to its characters, interactive gameplay elements [65] and/or even the music. The current mapping to polygons is insufficient to support these kind of fitness functions. Other mappings might be more suitable for fitness functions in games, for example graph-based structures [61] could be explored.

While the current probabilistic model is capable of optimising some fitness functions in the learning process, there is still a lot of room improvement. Other probabilistic graphical models, which could yield better results, are Bayesian networks or hidden Markov Models. In favour of keeping the methodology white-box and completely integrating the learning and generative process, I severely reduced the possible complexity of the graphical model. For future research, using more complex graphical models like neural networks might yield better results [88].

Instead of using a purely random sampler, the influence of an optimised sampler could be researched, for example an MCMC sampler. An optimised sampler will generate better content, therefore influencing the learning process and the content expressive range.

The learning process could be improved by not using existing learning algorithms, which are designed for standard machine learning problems like regression and density estimation, but rather implement a learning algorithm which is specifically tailored to increase the expressive range of a probabilistic model. This is similar to the exploration-vs-exploitation tradeoff mentioned in the related work chapter. Ideally the learning algorithm should not only search for optimal parameters but should also be encouraged to find parameters which result in higher variance in the probabilistic models.

In machine learning, training data can be pre-processed in many ways before applying learning algorithms on it, including the use of outlier removal, dimensionality reduction and feature extraction. Applying these in the learning process could improve the methodology. However, the data should never be irreversibly transformed. Only if an inverse of the transformation exists, a model trained on the transformed data can be sampled to generate artificial data which can be used as substitute for the original training data.

When optimising multiple fitness functions, machine learning techniques could also be used to detect fitness function which conflict or inversely complement each other. A possible path for future research could be the use of covariance analysis.

Even though the evaluation of expressiveness has been addressed, the issue is not completely solved. One promising approach would be to use more complex expressiveness evaluation for example with simulations, surrogate models or even human interaction. These could then be combined with machine learning techniques, like Bayesian optimisation, which are able to optimise expensive objective functions without access to derivatives with respect to its input and where the function is nonconvex and multimodal [101].

Bibliography

[1] entertainment software association, "Essential facts about the computer and video game industry," 2015.

[2] entertainment software association, "Annual report 2014," 2015. Consulted on 25-May-2016 via http://engineering.purdue.edu/ mark/puthesis.

[3] flemish games association, "Gaming in flanders," 2014.

[4] James Paul Gee, "Good Fit in Good Video Games: Components in a System," 2014.

[5] Y. Takatsuki, "Cost headache for game developers," 2007. Consulted on 25-May-2016 via http://news.bbc.co.uk/2/hi/business/7151961.stm.

[6] A. Iosup, "POGGI: Puzzle-based online games on grid infrastructures," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5704 LNCS, pp. 390–403, 2009.

[7] P. D. Groote, "Ik wil me niet meer verstoppen," 2015. Consulted on 25-May-2016 via http://www.tijd.be/ondernemen.

[8] G. Kelly and H. McCabe, "A survey of procedural techniques for city generation," *ITB Journal*, vol. 14, pp. 87–130, 2006.

[9] D. M. D. Carli, C. T. Pozzer, F. Bevilacqua, and V. Schetinger, "Procedural Generation of 3D Canyons," in *2014 27th SIBGRAPI Conference on Graphics, Patterns and Images*, pp. 103–110, IEEE, 2014.

[10] S. Dahlskog and J. Togelius, "A multi-level level generator," in *2014 IEEE Conference on Computational Intelligence and Games*, pp. 1–8, IEEE, 2014.

[11] D. Quick, "Generating Music Using Concepts from Schenkerian Analysis and Chord Spaces," 2010.

[12] M. Schwarz, "Advanced Procedural Modeling of Architecture," 2015.

[13] E. J. Hastings, R. K. Guha, L. Member, and K. O. Stanley, "Galactic Arms Race Video Game," *Computational Intelligence*, vol. 1, no. 4, pp. 245–263, 2009.

[14] K. Hartsook, A. Zook, S. Das, and M. O. Riedl, "Toward supporting stories with procedurally generated game worlds," *2011 IEEE Conference on Computational Intelligence and Games, CIG 2011*, pp. 297–304, 2011.

[15] B. Kybartas and C. Verbrugge, "Analysis of ReGEN as a Graph-Rewriting System for Quest Generation," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, pp. 228–242, jun 2014.

[16] G. Smith and J. Whitehead, "Analyzing the expressive range of a level generator," *Proceedings of the 2010 Workshop on Procedural Content Generation in Games PCGames 10*, 2010.

[17] W. L. Raffe, F. Zambetta, and X. Li, "Neuroevolution of content layout in the PCG: Angry bots video game," in *2013 IEEE Congress on Evolutionary Computation*, pp. 673–680, IEEE, 2013.

[18] A. M. Smith and M. Mateas, "Answer Set Programming for Procedural Content Generation: A Design Space Approach," *IEEE Trans. Comput. Intell. AI Games*, vol. 3, pp. 187–200, sep 2011.

[19] G. N. Yannakakis and J. Togelius, "Experience-Driven Procedural Content Generation," *IEEE Transactions on Affective Computing*, vol. 2, no. 3, pp. 147–161, 2011.

[20] C. a. Vanegas, I. Garcia-Dorado, D. G. Aliaga, B. Benes, and P. Waddell, "Inverse design of urban procedural models," *ACM Transactions on Graphics*, vol. 31, no. 6, 2012.

[21] D. M. De Carli, F. Bevilacqua, C. T. Pozzer, and M. C. D'Ornellas, "A survey of procedural content generation techniques suitable to game development," in *Brazilian Symposium on Games and Digital Entertainment, SBGAMES*, pp. 26–35, IEEE, 2011.

[22] L. Villapaz, "'gta 5' costs 265 million dollar to develop and market, making it the most expensive video game ever produced: Report," 2013. Consulted on 25-May-2016 via http://www.ibtimes.com/gta-5-costs-265-million-develop-market-making-it-most-expensive-video-game-ever-produced-report.

[23] C. Hosking, "Opinion: Stop dwelling on graphics and embrace procedural generation," 2013. Consulted on 25-May-2016 via http://www.polygon.com/2013/12/10/5192058/opinion-stop-dwelling-on-graphics-and-embrace-procedural-generation.

[24] J. Toglius, "Why academics and game industry don't collaborate on ai, and how we could improve the situation," 2014. Consulted on 25-May-2016 via http://togelius.blogspot.com/2014/10/why-academics-and-game-industry-dont.html.

[25] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup, "Procedural content generation for games," *ACM Trans. Multimed. Comput. Commun. Appl.*, vol. 9, pp. 1–22, 2013.

[26] G. Smith, "The Future of Procedural Content Generation in Games," *Experimental AI in Games Workshop at AIIDE*, 2014.

[27] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-Based Procedural Content Generation: A Taxonomy and Survey," *IEEE Trans. Comput. Intell. AI Games*, vol. 3, no. 3, pp. 172–186, 2011.

[28] D. Ritchie, "Controlling Procedural Modeling Programs with Stochastically-Ordered Sequential Monte Carlo," 2015.

[29] W. L. Raffe, F. Zambetta, X. Li, and K. O. Stanley, "Integrated Approach to Personalized Procedural Map Generation Using Evolutionary Algorithms," 2015.

[30] N. Sorenson, P. Pasquier, and S. DiPaola, "A Generic Approach to Challenge Modeling for the Procedural Creation of Video Game Levels," *IEEE Trans. Comput. Intell. AI Games*, vol. 3, pp. 229–244, sep 2011.

[31] J. O. Talton, Y. Lou, S. Lesser, J. Duke, R. Měch, and V. Koltun, "Metropolis procedural modeling," *ACM Trans. Graph.*, vol. 30, no. 2, pp. 1–14, 2011.

[32] N. Shaker, G. Smith, and G. Yannakakis, "Evaluating content generators," *Proced. Content Gener. Games A Textb. an Overv. Curr. Res.*, pp. 211–218, 2015.

[33] R. Smelik, T. Tutenel, K. J. de Kraker, and R. Bidarra, "Integrating procedural generation and manual editing of virtual worlds," *Proc. 2010 Work. Proced. Content Gener. Games*, pp. 1–8, 2010.

[34] R. M. Smelik, T. Tutenel, K. J. De Kraker, and R. Bidarra, "A declarative approach to procedural modeling of virtual worlds," *Comput. Graph.*, vol. 35, no. 2, pp. 352–363, 2011.

[35] J. Togelius, N. Shaker, and J. Dormans, "Grammars and L-systems with applications to vegetation and levels (DRAFT)," *Proced. Content Gener. Games A Textb. an Overv. Curr. Res.*, pp. 73–96, 2015.

[36] P. Prusinkiewicz and A. Lindenmayer, "The algorithmic beauty of plants," *Plant Sci.*, vol. 122, no. 1, pp. 109–110, 1997.

[37] L. Krecklau, D. Pavic, and L. Kobbelt, "Generalized use of non-terminal symbols for procedural modeling," *Comput. Graph. Forum*, vol. 29, no. 8, pp. 2291–2303, 2010.

[38] J. Dormans, "Adventures in level design: generating missions and spaces for action adventure games," *... Work. Proced. Content Gener. Games*, pp. 1–8, 2010.

[39] J. Talton, L. Yang, R. Kumar, M. Lim, N. Goodman, and R. Měch, "Learning design patterns with bayesian grammar induction," *Proc. 25th Annu. ACM Symp. User interface Softw. Technol. - UIST '12*, p. 63, 2012.

[40] P. B. Silva, P. Müller, R. Bidarra, and A. Coelho, "Node-Based Shape Grammar Representation and Editing," *Pcg*, pp. 1–8, 2013.

[41] M. Lipp, P. Wonka, and M. Wimmer, "Interactive visual editing of grammars for procedural architecture," *ACM Trans. Graph.*, vol. 27, no. 3, p. 1, 2008.

[42] P. B. Silva, E. Eisemann, R. Bidarra, and A. Coelho, "Procedural Content Graphs for Urban Modeling," *Int. J. Comput. Games Technol.*, vol. 2015, pp. 1–15, 2015.

[43] K. Compton, B. Filstrup, and M. Mateas, "Tracery : Approachable Story Grammar Authoring for Casual Users," *Pap. from 2014 AIIDE Work. Intell. Narrat. Technol. (7th INT, 2014)*, pp. 64–67, 2014.

[44] G. Smith, J. Whitehead, and M. Mateas, "Tanagra: A mixed-initiative level design tool," *FDG '10 - Proc. Fifth Int. Conf. Found. Digit. Games*, pp. 209–216, 2010.

[45] A. Liapis, G. N. Yannakakis, and J. Togelius, "Sentient sketchbook: Computer-aided game level authoring," *Proc. 8th Int. Conf. Found. Digit. Games (FDG 2013)*, pp. 213–220, 2013.

[46] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelb??ck, G. N. Yannakakis, and C. Grappiolo, "Controllable procedural map generation via multiobjective evolution," *Genet. Program. Evolvable Mach.*, vol. 14, no. 2, pp. 245–277, 2013.

[47] R. M. Smelik, T. Tutenel, K. J. De Kraker, and R. Bidarra, "A declarative approach to procedural modeling of virtual worlds," *Comput. Graph.*, vol. 35, no. 2, pp. 352–363, 2011.

[48] D. Plemenos and G. Miaoulis, *Visual Complexity and Intelligent Computer Graphics Techniques Enhancements*. 2009.

[49] M. Bennett, "Semantic content generation framework for game worlds," in *2014 6th Int. Conf. Games Virtual Worlds Serious Appl. VS-GAMES 2014*, vol. 35, pp. 352–363, 2015.

[50] F. P. J. Brooks, "No silver bullet-essence and accidents of software engineering," *Proc. IFIP Tenth World Comput. Conf.*, pp. 1069–1076, 1986.

[51] S. Snodgrass, "Probabilistic Foundations for Procedural Level Generation," pp. 18–21, 2014.

[52] C. McGuinness, "Statistical analyses of representation choice in level generation," in *2012 IEEE Conf. Comput. Intell. Games, CIG 2012*, pp. 312–319, 2012.

[53] J. Togelius and G. Yannakakis, "Search-based procedural content generation," *Appl. ...*, pp. 1–10, 2010.

[54] D. Ashlock, S. Risi, and J. Togelius, "09. Representations for search-based methods," *Proced. Content Gener. Games A Textb. an Overv. Curr. Res.*, pp. 153–173, 2015.

[55] F. Rothlauf, *Representations for genetic and evolutionary algorithms.* Springer Verlag, Berling, Heidelberg, Netherlands, 2006.

[56] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelbäck, and G. N. Yannakakis, "Multiobjective exploration of the StarCraft map space," in *Proc. 2010 IEEE Conf. Comput. Intell. Games, CIG2010*, pp. 265–272, 2010.

[57] M. Shaker, N. Shaker, J. Togelius, and M. Abou-Zleikha, "A progressive approach to content generation," in *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9028, pp. 381–393, 2015.

[58] M. O'Neill and C. Ryan, "Grammatical evolution," *IEEE Trans. Evol. Comput.*, vol. 5, no. 4, pp. 349–358, 2001.

[59] M. Shaker, M. H. Sarhan, O. A. Naameh, N. Shaker, and J. Togelius, "Automatic generation and analysis of physics-based puzzle games," in *IEEE Conf. Comput. Intell. Games, CIG*, 2013.

[60] G. Minella, R. Ruiz, and M. Ciavotta, "A review and evaluation of multiobjective algorithms for the flowshop scheduling problem," 2008.

[61] R. Lara-Cabrera, C. Cotta, and A. J. Fernández-Leiva, "A self-adaptive evolutionary approach to the evolution of aesthetic maps for a RTS game," 2014.

[62] M. Preuss, A. Liapis, and J. Togelius, "Searching for good and diverse game levels," in *IEEE Conf. Comput. Intell. Games, CIG*, pp. 1–8, 2014.

[63] T. Tutenel, R. M. Smelik, R. Lopes, K. J. De Kraker, and R. Bidarra, "Generating consistent buildings: A semantic approach for integrating procedural techniques," in *IEEE Trans. Comput. Intell. AI Games*, vol. 3, pp. 274–288, 2011.

[64] A. Uriarte and S. Ontanón, "Psmage: Balanced map generation for starcraft," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pp. 1–8, IEEE, 2013.

[65] M. Cook and S. Colton, "Multi-faceted evolution of simple arcade games," in *2011 IEEE Conf. Comput. Intell. Games, CIG 2011*, pp. 289–296, 2011.

[66] M. Cook, S. Colton, and J. Gow, "Initial results from co-operative co-evolution for automated platformer design," in *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7248 LNCS, pp. 194–203, 2012.

[67] S. Risi and J. Togelius, "Neuroevolution in Games: State of the Art and Open Challenges," *CoRR*, vol. abs/1410.7, no. c, pp. 1–19, 2014.

[68] K. O. Stanley, "Compositional pattern producing networks: A novel abstraction of development," *Genet. Program. Evolvable Mach.*, vol. 8, no. 2, pp. 131–162, 2007.

[69] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies.," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, 2002.

[70] M. Kerssemakers, J. Tuxen, J. Togelius, and G. N. Yannakakis, "A procedural procedural level generator generator," in *2012 IEEE Conf. Comput. Intell. Games, CIG 2012*, pp. 335–341, 2012.

[71] Y.-T. Yeh, L. Yang, M. Watson, N. D. Goodman, and P. Hanrahan, "Synthesizing open worlds with constraints using locally annealed reversible jump MCMC," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 1–11, 2012.

[72] M. Fisher, D. Ritchie, M. Savva, T. Funkhouser, and P. Hanrahan, "Example-based synthesis of 3D object arrangements," *ACM Trans. Graph.*, vol. 31, no. 6, p. 1, 2012.

[73] P. Merrell, E. Schkufza, Z. Li, M. Agrawala, and V. Koltun, "Interactive furniture layout using interior design guidelines," *ACM Trans. Graph.*, vol. 30, no. 4, p. 1, 2011.

[74] M. M. Kelsey Kruse, *Interior Graphic Standards.* JohnWiley and Sons, 2003.

[75] A. Hertzmann, "Machine learning for computer graphics: A manifesto and tutorial," in *Proc. - Pacific Conf. Comput. Graph. Appl.*, vol. 2003-Janua, pp. 22–36, 2003.

[76] E. Brochu, T. Brochu, and N. D. Freitas, "A Bayesian Interactive Optimization Approach to Procedural Animation Design," *Symp. Comput. Animat.*, pp. 103–12, 2010.

[77] L. V. Carvalho, Á. V. M. Moreira, V. V. Filho, M. T. C. F. Albuquerque, and G. L. Ramalho, "A Generic Framework for Procedural Generation of Gameplay Sessions," pp. 203–210, 2013.

[78] Y. Jiang, M. Lim, and A. Saxena, "Learning Object Arrangements in 3D Scenes using Human Context," *Proc. 29th Int. Conf. Mach. Learn.*, pp. 1543–1550, 2012.

[79] P. Merrell, E. Schkufza, and V. Koltun, "Computer-generated residential building layouts," *ACM Trans. Graph.*, vol. 29, no. 6, p. 1, 2010.

[80] J. O. Talton, D. Gibson, L. Yang, P. Hanrahan, and V. Koltun, "Exploratory modeling with collaborative design spaces," *ACM Trans. Graph.*, vol. 28, no. 5, p. 1, 2009.

[81] D. R. Jones, "A Taxonomy of Global Optimization Methods Based on Response Surfaces," *J. Glob. Optim.*, vol. 21, no. 4, pp. 345–383, 2001.

[82] M. Kurek, T. Becker, and W. Luk, "Parametric optimization of reconfigurable designs using machine learning," *Reconfigurable Comput. Archit. ...*, pp. 134–145, 2013.

[83] M. Li, S. Yang, and X. Liu, "Shift-based density estimation for pareto-based algorithms in many-objective optimization," *IEEE Trans. Evol. Comput.*, vol. 18, no. 3, pp. 348–365, 2014.

[84] S. Calinon, A. Pervez, and D. G. Caldwell, "Multi-optima policy search with adaptive Gaussian mixture model," 2013.

[85] P. Feliot, J. Bect, and E. Vazquez, "A Bayesian approach to constrained multi-objective optimization," in *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8994, pp. 256–261, 2015.

[86] H. Karshenas, R. Santana, C. Bielza, and P. Larranaga, "Multi-objective Estimation of Distribution Algorithm Based on Joint Modeling of Objectives and Variables," *IEEE Trans. Evol. Comput.*, no. c, pp. 1–1, 2013.

[87] P. A. N. Bosman and D. Thierens, "Multi objective optimization with diversity preserving mixture based iterated density estimation evolutionary algorithms," no. December 2005, 2005.

[88] A. Dosovitskiy, J. T. Springenberg, and T. Brox, "Learning-To-Generate-Chairs-With-Convolutional-Neural-Networks," in *IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 1538–1546, 2015.

[89] J. Amores, "Multiple instance classification: Review, taxonomy and comparative study," *Artificial Intelligence*, vol. 201, pp. 81–105, 2013.

[90] N. Li, X. Zhao, D. Li, J. Wang, and B. Bai, "Object tracking with multiple instance learning and gaussian mixture model," *Journal of Information and Computational Science*, vol. 12, no. 11, pp. 4465–4477, 2015.

[91] J. S. De Bonet, C. L. Isbell, P. Viola, *et al.*, "Mimic: Finding optima by estimating probability densities," *Advances in neural information processing systems*, pp. 424–430, 1997.

[92] D. C. Kessler, J. A. Taylor, and D. B. Dunson, "Learning phenotype densities conditional on many interacting predictors," *Bioinformatics*, vol. 30, no. 11, pp. 1562–1568, 2014.

[93] B. E. Hansen, "Nonparametric conditional density estimation," *Unpublished manuscript*, 2004.

[94] M. L. Eaton and M. Eaton, *Multivariate statistics: a vector space approach.* No. 3, Wiley New York, 1983.

[95] T. B. Schön and F. Lindsten, "Manipulating the multivariate gaussian density," tech. rep., Technical report, Linkoeping University, 2011. http://www. rt. isy. liu. se/schon/Publications/SchonL2011. pdf, 2011.

[96] J. W. Harris and H. Stöcker, *Handbook of mathematics and computational science.* Springer Science & Business Media, 1998.

[97] "Pomegranate, probabilistic modelling library for Python."

[98] S. Ghahramani, *Fundamentals of Probability with stochastic process.* Pearson Education, 1994.

[99] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.

[100] T. S. Nielsen, G. a. B. Barros, J. Togelius, and M. J. Nelson, "General Video Game Evaluation Using Relative Algorithm Performance Profiles," *Proc. 18th Conf. Appl. Evol. Comput.*, pp. 369–380, 2015.

[101] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. D. Freitas, "Taking the Human Out of the Loop : A Review of Bayesian Optimization," *RLTutor*, vol. 104, no. 1, pp. 1–24, 2015.

[102] Newzoo, "Global games market will grow 9.4% to $ 91.5bn in 2015," 2013. Consulted on 25-May-2016 via http://www.newzoo.com/insights/global-games-market-will-grow-9-4-to-91-5bn-in-2015/.

[103] C. Hosking, "Opinion: Stop dwelling on graphics and embrace procedural generation," 2013. Consulted on 25-May-2016 via http://www.polygon.com/2013/12/10/5192058/opinion-stop-dwelling-on-graphics-and-embrace-procedural-generation.

[104] L. Villapaz, "Gta 5 costs 265 million dollar to develop and market, making it the most expensive video game ever produced: Report," 2013. Consulted on 25-May-2016 via http://www.ibtimes.com/gta-5-costs-265-million-develop-market-making-it-most-expensive-video-game-ever-produced-report.

[105] B. Lake and J. Tenenbaum, "Computational Creativity : Generating new objects with a hierarchical Bayesian model," no. 2013, p. 3325, 2014.

List of Figures

List of Tables