NTNU
Norwegian University of
Science and Technology

Distribution-based Grammatical Evolution of L-system Plants

Magnus Bjerke Vik

Applied Computer Science
Submission date: December 2017
Supervisor: Mariusz Nowostawski, IDI

Norwegian University of Science and Technology
Department of Computer Science

Preface

This is a Master's thesis in Applied Computer Science at the Norwegian University of Science and Technology that was carried out during the spring semester of 2017.

It is based on an idea of a virtual world where people can create and experience a garden by growing a wide variety of random generated plants that may be recognizable or unexpected, but always aesthetically pleasing. They can then crossbreed these plants into new surprising plants that have distinct features from both of the parent plants, letting the user's creativity go wild.

The reader of this thesis will benefit of general knowledge in computer science and computer programming. Additional knowledge in evolutionary computing will be helpful, but not strictly necessary.

15-12-2017

Acknowledgment

I would like to thank my supervisor, Mariusz Nowostawski, for helping me with his insight, interesting ideas and discussions. I would also like to thank my girlfriend, Merete Nilssen Ringen, for her support and grammatical fixes. Finally, I would like to thank my classmates for making the process of writing the thesis a bit more fun.

M.B.V.

Abstract

This thesis explores how to improve L-system plant generation using evolutionary algorithms (EAs). The plants should be aesthetically pleasing, varied, and have the ability to be combined with other plants.

Previous work has shown that simple D0L-systems, PD0L-systems and PDIL-systems can be evolved using EAs both autonomously and interactively. The L-system grammar used is simple, restricting the possible solutions. Additionally, often only parts of the grammar is represented in the genotype that is used in the EA, thus limiting the evolution.

More complex L-systems are required to generate aesthetically pleasing and varied plants, but this also complicates the generator and therefore also the evolution. To mitigate this issue, distribution-based grammatical evolution of L-systems (DGEL) is introduced and implemented. It is based on grammatical evolution (GE) of L-systems, but introduces a grammar distribution to control what parts of the grammar that should have a higher priority. Simulated annealing (SA) is used to optimize this grammar distribution so that DGEL can produce well performing plants in a more efficient manner. Additionally, using different optimized grammar distributions could allow for a larger variety of plants, while still keeping the quality up. Fitness metrics used were both adapted from the reviewed literature and created from scratch to assess the pleasingness of the generated L-system plants.

DGEL was evaluated in three parts: GE, SA and fitness function. Plain GE was tested against random brute-force to see if it has any benefit. SAs progress and produced grammar distribution was studied, and its performance was compared using both GE and random brute-force. The fitness metric was compared to human evaluations of generated L-system plants through analytical hierarchy process (AHP) and rank correlation statistics. Additionally, it was analyzed what factors humans think are important for distinguishing good from bad plants. GE was found to be superior to brute-force. SA was found to improve the efficiency of both brute-force and GE. Finally, the fitness function was found to not match the human evaluation, but a small correlation was still present.

The findings suggest that DGEL can improve the efficiency of generating L-system plants in a complex space. In a more general sense, the findings suggest that accompanying an EA with a probability distribution, and optimizing it using optimization techniques, can improve its efficiency in complex spaces, allowing for faster searches and more varied solutions.

Contents

List of Figures

List of Tables

List of Listings

Glossary

branch base A point on a branch with at least two child branch segments [1]. 22, 24–26, 29, 30, 32

branch segment An edge in an interpreted L-system [1]. 19, 20, 22, 24, 26, 28–32

child segment A branch segment following a parent segment in direction from the root. 29, 30, 32

D0L-system Discrete context free L-systems. v

internode segment "A branch segment followed by at least one more segment in some path" [1]. 32

L-system Lindenmayer systems. v, 1–5, 7–17, 19, 22, 24–29, 33–35, 37, 39, 41, 48–50, 54, 72, 73, 75–79, 81–84, 95

PD0L-system Parametric discrete context free L-systems. v, 81

PDIL-system Parametric discrete context sensitive L-systems. v, 81

straight segment The child segment that continues the branch (as opposed to child segments branching into new branches) [1]. 32

Acronyms

AHP analytical hierarchy process. v, 55, 56, 58, 60, 62, 73, 82

ANN artificial neural network. 2, 4

DGEL distribution-based grammatical evolution of L-systems. v, 17–24, 26–29, 31, 33, 34, 39–41, 54, 55, 71, 74, 76–79, 81–84, 95

EA evolutionary algorithm. v, 1–4, 9–12, 73, 78, 83

GA genetic algorithm. 10, 11, 13, 15, 27, 37, 39, 81, 84

GE grammatical evolution. v, 10–12, 15, 17, 19, 24, 27, 33–35, 37, 39–45, 48, 50, 51, 55, 60, 71, 73, 76–79, 81, 82, 84

GP genetic programming. 10–12, 15, 81

PCG procedural content generation. 4, 78

SA simulated annealing. v, 17, 26, 27, 33, 35–37, 39, 41, 42, 46–51, 53, 71–73, 77–79, 82, 83

VCS version control system. 35

1 Introduction

1.1 Topic covered by the project

This project covers the topic of evolutionary algorithm (EA) which is part of evolutionary computing and even more broadly artificial intelligence. More specifically, it covers the use of EA to evolve L-systems into good graphical models of aesthetically pleasing plants.

L-systems were originally conceived as a mathematical model for cellular interactions in development [2, 3]. From this, geometric interpretations of L-systems have been created so that they can be used to model plants [1]. An L-system is a rewriting system, meaning that it rewrites itself based on a set of rules defined by a grammar. A simple example is an L-system with the rules $a \rightarrow b$ and $b \rightarrow ba$, and the axiom a. Based on the rules, the a axiom is first rewritten to b. Then b is rewritten to ba, ba is rewritten to bab, which is rewritten to $babba$, and so on.

These rewritten strings do not mean anything on their own, and that is where the interpretation comes in. If we interpret b as rotating 45° left and a as drawing a line 1 cm forward, the L-system would model a growing structure, as visualized in Figure 1. If the alphabet is expanded from only a and b, and more interpretations are added, L-systems can model complex structures in both 2D and 3D, such as plants.

How do you then create L-systems that model the complex structures you desire? They could be hand crafted by manually defining the rules, but this is difficult for non-experts and require significant amounts of experimentation and analysis of the desired structure [4]. Additionally, what if an exact structure is not desired, but rather a structure that fulfills some requirement, for example being aesthetically pleasing? This is where EA is useful.

EA is not directly connected to L-systems, but it is a tool that can be used to

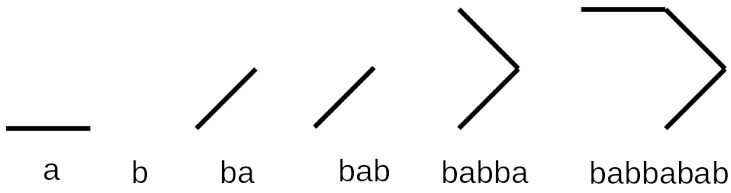


Figure 1: Example L-system being rewritten and interpreted

evolve data structures, such as L-systems. EAs generate an initial population of structures, then repeatedly evaluate their quality, select structures to reproduce, reproduce them and replaces the population with the new structures until they are satisfied [5]. By repeatedly selecting good individuals and reproducing them, EA may find solutions to problems that perform exceptionally well. It is used in many fields, and has many applications. For example, EA has been used to find weights for an artificial neural network (ANN) that is used in a virtual robot, it has been used to optimize the fuel consumption of stoves, and is being used for modeling complex or noisy data [5]. Finally, it has of course been used to evolve L-systems, both simple fractal structures and 2D and 3D plants [6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18].

1.2 Keywords

- L-system
- plants
- artificial intelligence
- evolutionary algorithm
- genetic algorithm
- grammatical evolution
- procedural content generation

1.3 Problem description

Real plants can be bred to create offspring with desirable features from multiple species. For example, many of the species used in agriculture have gone through selection over several years to yield plants that produce a bigger quantity of food and are resistant to diseases and harsh environments. Some plants were domesticated several thousand years ago, for example the popular plant Maize has indication that it was domesticated 8700 years ago [19], while crossing between different species started in the 18th century [20]. This can be a complex and long process, taking several hundreds of years with gradual improvements, and many combinations do not even work. [21, 20]

To make this process more fun and interesting, a virtual world where people can breed plants could be created. In this world, breeding is not limited like it is in the real world. Plants could be combined in ways never thought to be possible. For example, a user may like both apple trees and Venus flytraps, so they want to combine them into a tree with Venus flytrap mouths of a size that could eat humans. The problem is that randomly crossing the representation of different plants may not produce meaningful or interesting offspring, but rather random creatures that do not satisfy the user.

To be able to create meaningful crossovers between virtual plants, a baseline requirement is to have something that models the plants and something that generates the plants. The plant model is a baseline requirement of the system, as no generator or crossover would work without this, and a generator is required to generate the parent plants. Additionally the generator could be used to cross the plants into something the user is satisfied with.

Plants that are to be bred should be varied in their shape and features, like real plants are, otherwise interesting combinations can not be made. Additionally, since the breeding should be able to create combinations that are impossible in the real world, the variation should be even larger, allowing for plants not found in nature. As a baseline the plants should be aesthetically pleasing, otherwise the user of a virtual environment with these plants would most likely be dissatisfied. This is especially important in video games where a user may quit if they do not enjoy playing it. Finally, if the plants should be crossed, their model should be represented in such a way that genes can be interchanged in a meaningful way that retains the properties of the plants.

The current research on evolving L-systems using EA uses strict restrictions on the grammar which limits the variation in the plant, in addition to its potential for being aesthetically pleasing. Thus a less restricting grammar is required, but this will increase the complexity of the problem, making it more difficult to produce meaningful and pleasing plants.

In summary the problem to be studied is how to generate aesthetically pleasing and varied L-system plants that could be used for crossbreeding, with a particular focus on how to handle complex grammars.

1.4 Justification, motivation and benefit

As is apparent from Chapter 2, currently EA has only been applied to restricted L-systems or only parts of L-systems. The research often only studies the simplest types of L-systems [6, 7, 8, 9, 10, 11, 12], while sometimes also the more versatile parametric L-systems [13, 14, 15]. In both cases, the L-systems are usually based on assumptions about the size of the alphabet, string lengths, number of productions and branching depths. This restricts the potential power of L-systems, thus also restricting the potential of generating complex and intricate structures, for example aesthetically pleasing and varied plants. This project explores a more expressive L-system as well as techniques to guide the generation of good solutions with it. While we do not explore the most flexible forms of L-systems, the techniques explored can potentially be applied to these as well.

These benefits are not limited to only L-systems, but can be applied to any problems that use EA and grammars, for example computer programs, with some

modifications. Looking at the wide specter of applications of EA [5], it is apparent that an improvement in EA techniques will have indirect beneficial effects for humanity.

Because L-systems are used in procedural content generation (PCG) [22], computer graphics, video games and film can benefit from this project. PCG is defined as "the algorithmic creation of game content with limited or indirect user input" [23], and has been used in video games since the 1980s [24]. It can remove the need for artists to lower costs, it may support the artists such that they may be more efficient and create better quality content, it can enable the creation of completely new games, it can adapt the content to the user's abilities or needs, it may allow for more creative solutions than what humans are able to create, and it may help understand design [24].

L-systems and formal grammars in general can be used to generate game content such as plants, missions, spaces and levels [22]. By studying how L-systems with EA can be used to generate both aesthetically pleasing and varied plants, we will get a better understanding of what factors are important in aesthetic plants, how a wide specter of different L-systems can be generated and how to handle complex spaces in a more efficient manner. Therefore, this project can improve the techniques used in PCG to help the game, graphics and film industries in creating better content and in the end create better experiences for consumers.

With an understanding of how to measure to what degree an L-system plant is aesthetically pleasing, an understanding on how L-systems can be genetically represented and combined, and an understanding of how to generate L-systems in a large parameter space, a basis for combining L-system plants into good offspring is created. This can be directly useful for PCG, as new game concepts or game mechanics of interactions with plants can be created. Taking this a step further, the findings could be adapted to other types of content, such as game levels where two levels with different concepts should be combined into a new level that borrows concepts from both or combines the concepts into new ones. In an even broader scope, because EA is being used, the techniques may be adapted to other problems, such as ANN, computer programs or architecture to search complex spaces or combine two solutions with some desired properties each into one solution with the properties combined.

1.5 Research questions

The problem described can be summarized in a single research question: How can we generate L-system plants that are aesthetically pleasing, varied and could be used to create offspring similar to its parents? The base problem to be solved in this question is how to generate plants, and is accompanied with three require-

ments: being aesthetically pleasing, varied and usable for cross-breeding. Thus the research question is split into three sub-questions that aim to tackle the three requirements, such that the main research question may be answered. The questions are listed below.

RQ0 How can we generate L-systems plants that are aesthetically pleasing, varied and could be used to create offspring similar to its parents?

- **RQ1** What L-systems models are appropriate to represent plants both found in nature and not found in nature, and that could be combined into new plants?
- **RQ2** How can aesthetically pleasing L-systems plants be generated?
- **RQ3** How can varied L-systems plants be generated?

1.6 Contribution

- A literature review of plant L-system representations and evolutionary algorithms used on them.
- A method of generating well-performing plants from a large parameter space.
- An analysis of metrics that evaluate how aesthetically pleasing an L-system plant is, and important factors as indicated by humans.

2 L-system Representation and Evolution

2.1 L-systems

An L-system consists of an alphabet, an axiom and a set of production rules. It is a parallel rewrite system where each letter in the axiom word is rewritten independently based on the production rules. This happens iteratively, where a new word replaces the previous word each iteration, as explained in Section 1.1. A way to draw the plant based on the L-system has to be applied to visualize it. Prusinkiewicz and Lindenmayer summarize multiple research papers into a comprehensive book about L-systems [1].

There are also multiple types of L-systems, including discrete, stochastic, context sensitive and parametric [1]. All of these can be combined into one L-system. A stochastic, context sensitive and parametric L-system (PS2L-System) can model all of the other types of L-system by having 100% probabilities (making it discrete), no contexts or no parameters, and is thus the most flexible representation.

Discrete context-free L-systems (D0L-systems) are the simplest L-systems, and can be created using edge rewriting, node rewriting, or both [1]. In these systems, there exists one production rule per letter in the alphabet. By default the productions will produce the same letter as the input (identity).

Stochastic L-systems add a randomness to the generation of the plants. Each letter in the alphabet may have multiple productions, each with a probability of being selected [1]. This may simulate how different instances of a plant species may look slightly different, thus making the plants look less synthetic when seen together with other plants of the same species.

Context sensitive L-systems add a context to the production rule. The context can be on either left, right or both sides of the letter. A production will only be used if the context matches the surrounding letters in the word. With this, signal propagation can be simulated either upwards or downwards through the plant [1], and more complex plants may be generated.

Parametric L-systems adds parameters to the letters in the words, and conditional rules [1]. The main benefit of using parametric L-systems, is that it can work with rational numbers rather than only integers. For example, with non-parametric L-systems, a growing segment can be modeled with the rule $\mathtt{F} \rightarrow \mathtt{FF}$, or any number of `F` in the successor (edge rewriting). With this rule the number of `F`s will double—doubling the length of the branch—with each iteration. This means that

the length of a branch can only double with each iteration. For some plants, this is enough, but to model a larger variety of plants, rational numbers are required. With parametric L-systems, the same rule would be $F(l) \rightarrow F(l * 2)$, where l is the length of the segment. But now it is also possible to grow it at another rate, e.g. $F(l) \rightarrow F(l * 1.2)$. Prusinkiewicz and Lindenmayer describe more of the benefits of parametric L-systems [1].

Prusinkiewicz describe L-systems as having three levels of model specification: partial L-systems, L-system schemata, and complete L-systems [1]. The three different levels go from more abstract to more concrete, where partial L-systems specify which structures may result into which new structures (e.g. bud becomes a flower), schemata specifies when the different structure switches happen (e.g. when the bud becomes a flower), and complete systems specify the geometry of the plant to be visualized (e.g. how the bud and the flower should look). Different methods exist for each level, and which method to use will depend on the type of plant that should be generated.

L-system schemata is of particular interest because there exists multiple methods to use. The event of a structure resulting into a new structure is called a *developmental switch* [1]. The timing of the switches need to be controlled by a control mechanism in the system. There are two classes of these mechanisms: lineage and interaction. Lineage mechanisms are transferring information from a module to a descendant, while interaction mechanisms exchange information between cells.

Prusinkiewicz and Lindenmayer describe some of the mechanisms used. The stochastic mechanism uses a stochastic L-system to apply probabilities to developmental switches. A table L-system has multiple tables with different rules that can be applied depending on some external factor. For example, one table can represent summer and another can represent winter, making different switches happen at different seasons. The delay mechanism can delay the developmental switch by a specified number of iterations, for example making a flower bud bloom after a certain amount of iterations. Accumulation of components uses parametric L-systems to accumulate some parameter until it reaches a threshold causing the switch to happen. Parametric L-systems can also be used to control development switches with signals, where a signal travels either upwards or downwards through the plant. [1]

A popular way to render complete L-systems is the turtle interpretation, where a cursor (the turtle) follows instructions to draw lines, and the alphabet is the set of instructions to use [1]. Turtle interpretation is in its original form simple and can only draw lines on a 2D image, but it has been extended to draw realistic looking plants in 3D [1]. To use the turtle interpretation, extra parameters describing how much the L-system should be expanded and how it should be drawn has to be

defined [1]. For example, the number of rewrite iterations, the branching angle, the segment length, and the segment width have to be specified. This may vary depending on the plant to be rendered, as some plants might require a more complex set of drawing instructions.

2.2 Genetically Evolving L-systems

Genetic evolution involves crossing and mutating genes from parents to create offspring. Thus, the research field of L-system genetic evolution is a good starting point for finding out both how to generate L-systems and combine them. A set of research papers published between 1995 and 2013 was studied to see what different techniques have been used.

To get a better overview of the EA techniques used for L-systems, the EA has been split into five parts: model, representation, generation, selection, and genetic operators. The *model* and *representation* are related and sometimes almost equal. The difference is that the model describes how the plant is modeled, while the representation may be a small part of the model or the model represented in a different format, and is the part which is evolved by the algorithm. *Generation* is how the representations are generated, e.g. for the initial population, or when replacing parts of an L-system with a new part. *Selection* is the part of selecting individuals to be used for creating the next generation. *Genetic operators* are used on the selected individuals to modify the representation either by doing a crossover between parents, mutating an offspring, or other.

2.2.1 Model

Discrete context-free L-systems (D0L-systems) are the simplest form of L-systems and they are commonly used for EA [6, 7, 8, 9, 10, 11, 12]. Parametric discrete context-free L-systems (PD0L-systems) have also been prominently used in the literature [13, 14, 15]. Parametric discrete context-sensitive L-systems (PDIL-systems) have been used in Jacob's further work by extending the PD0L-system version [16, 17, 18]. Other types of L-systems, such as stochastic, timed, table, environmentally-sensitive and open L-systems have not been used for EA in the reviewed literature. Hornby and Pollack argue that stochastic L-systems are hard to use for EA because they are non-deterministic [15]. Additionally, they argue that context-sensitive L-systems are not as powerful as parametric L-systems, and should therefore not be used. They use PD0L-systems in their own research, but it is worth noting that they evolve moving creatures—not plants.

All of the above-mentioned L-systems are bracketed L-systems, i.e. they include stack operators (`[` and `]`) in their alphabets to push and pop the current interpretation state, which creates branches in the structure. Additionally, Hornby and

Pollack use repetition brackets ({ and }) with an integer parameter to specify how many times the content inside the brackets should be repeated [15].

The L-system type is only the first part of the model required to model a plant. The second and final part is the interpretation of the L-system into a visualized model. Even though in the literature they only use three types of L-systems (D0L, PD0L and PDIL), they use different ways of interpreting it.

Both Mock, Ochoa, and Beaumont and Stepney visualize the L-systems in 2D with branching, but while Mock uses node rewriting [6], Ochoa and also Beaumont and Stepney use edge rewriting [7, 11], which results in different types of plants. The remaining authors use 3D visualization with branching [13, 10], though Jacob adds the concepts of sprout, stalk leaf and bloom [16], and Ebner et al. only add leaves [8, 9].

2.2.2 Representation

The representation used for EA also varies. In the literature, there have been three main EA methods: genetic programming (GP), genetic algorithm (GA) and grammatical evolution (GE). Jacob and Ochoa use the GP methods [13, 16, 7], both based on techniques introduced by Koza [25]. Ochoa uses a simple version of GP, where representation only consists of one single rule successor [7]. The successor is divided into a hierarchy based on the branching in the string, such that top-level letters are on the top, and each stack below is one level down. In their paper they only use successors with one level of branching.

Jacob represents the whole L-system as a tree structure, such as a GP method commonly does [13]. The tree structure starts with the L-system root at the top, then with the axiom and rules as children, then with each rule as a child of rules. The axiom has one child for each letter, and each letter has a child for the parameter. Each rule have a left and right side, where the left is the predecessor and the right is the successor. The left node is composed the same way as the axiom, but without parameters. The right node is the same, but it can have parameters and also stacks for branching that follow a tree structure. It can practically represent any PD0L-system this way, but without conditions and with a limited set of letters. This could be extended.

GA is the second major method used in the literature. The main difference between GA and GP is that GA represents the genotype as a flat string of elements, while GP represents it as a tree structure. Both Mock, Ochoa and Corchado et al. use only a single rule successor to represent the genotype [6, 7, 12]. The successor is a plain string of symbols, including letters and the branching operators. While Ochoa only uses node rewriting [7], Mock and Corchado et al. only use edge rewriting [6, 12].

Ebner et al. represent the genotype as a set of production rules with one edge rewriting rule and 0–26 node rewriting rules, each with individual nodes ranging from A–Z [8, 9]. The predecessor is fixed based on the rule number. For example, the first rule predecessor is always `f`, while the second and third predecessors would be `A` and `B`. This means that the genotype is a set of strings (successors), instead of one single string. This would allow the GA to search a bigger space compared to Mock and Ochoa's genotypes.

Hornby and Pollack take this further and represent the genotype as a fixed number of production rules composed of a predecessor and successor [15]. In this case, the genotype has as set of pairs of strings, which would allow the GA to search a even bigger space.

Finally, Ashlock et al. encodes the L-system as a string of real numbers [10]. Some of the numbers encoded are used to index a table of a fixed set of L-system rules, and some are used directly in the L-system interpretation step. This enables the L-system and its interpretation to be tackled as one single parameter optimization problem. None of the previous genotypes presented represented the L-system interpretation, though Ebner et al. and Hornby and Pollock used PD0L-systems where some interpretation parameters can be moved into the L-system production strings. Even though the encoded genotype can affect more aspects of the L-system (rules and interpretation), it is more limited because of its fixed set of rules that are used. Thus it can search a smaller space than Hornby and Pollack's representation.

GE is the final and latest method used in the literature. It is "an [EA] that can evolve complete programs in an arbitrary language using a variable-length binary string" [26]. It was introduced for L-systems by Ortega et al. to evolve a D0L-system into fractal curves [27]. Beaumont and Stepney transferred this to generating D0L-system 2D plants to match a drawing of a plant [11]. Interestingly, L-systems themselves are grammar that produce plants, but Beaumont and Stepney's work define a grammar that produces the L-systems, which is then used to produce the plant. They represent the genome as a string of numbers that index into a Backus Naur Form (BNF) grammar in order to build the L-system. In this sense the representation is similar to that of Ashlock [10], but with integers instead of real numbers and a different way of mapping from numbers to the L-system. It is also similar to Jacob's GP representation because it uses a pool of expressions that could be compared to a BNF. When undergoing evolution, the genotype (string of numbers) will be modified by a GA. Thus the GE method is strongly related to GA.

In addition to the common mutation and crossover operators used in GA, GE uses duplication and prune operators because of their benefits in biology. The duplication operator picks a series of genes, copies them and pastes them at the end of the chromosome. If the original genes are mutated in such a way that they are

no longer functional, the duplicated genes can be used as a backup. The original genes may also be mutated in a beneficial way without removing their old functions. Finally, increased presence of the same genes may have biological effects. The prune operator removes unused genes, and therefore works well with the duplication operator that adds more and more genes. [28]

Beaumont and Stepney use a limited search space for the GE method. Their alphabet only consists of `F`, `+` and `-`, their axiom is always `F`, and only `F` can be the production predecessor, i.e. it is an edge rewriting L-system. They also have a fixed number of iterations and rotation angle. This can be expanded, and they did some experiments to try this. `+` and `-` was added as predecessor letters, and the axiom was allowed to contain any symbols and have any length. This expanded the search space, making the algorithm find fewer very bad solutions, but also far fewer good solutions. Finally, they added a second terminal symbol `X` in addition to `F`, allowing for both edge and node rewriting L-systems. This further expanded the search space, but required a significantly larger population size and more generations, making the computation time significantly longer. The search space for this GE representation is smaller than some of the other methods because of its use of non-parametric L-systems and limited set of symbols, but it could be expanded to include this.

In addition, there is a non-EA method introduced by Vanak that is still relevant, called XL-system [14]. Rather than evolving L-systems, the XL-system is another form of L-systems where two or more PD0L-systems are combined with an X-machine into an XL-system. The original L-systems are not modified in any way, but rather kept as two states in the X-machine. The XL-system is used as a normal L-system, running a specified amount of derivations to rewrite an axiom. The difference is that the XL-system switches between the contained L-systems based on specified conditions while iterating. The active L-system is the one switched to, and is the one that will be used for rewriting the axiom in the current derivation. This means that it for example can alternate between the productions of two L-systems every derivation, creating a phenotype with properties from both L-systems.

The benefit of XL-systems is that they are a lossless combination of L-systems, compared to the EA methods presented where properties of the L-systems may be lost. The drawback is that the original L-systems have to be stored, causing an XL-system combining several generations of L-systems to be a large and complex structure.

2.2.3 Generation

There are multiple ways of generating individuals, depending on the representation used. When using a GP representation, one option is to randomly attach subex-

pressions based on pattern matching [13]. This works by using a tree structure that describes the pattern of the expressions that can be attached to child nodes. One of the expressions that match a pattern is randomly selected and attached to the node. Then it continues recursively until it reaches a terminal node.

With GA, it depends on the specific representation. To generate individuals with Mock's representation, they first select a random string length in some range for the successor, then fill the string with random characters from a set and finally repairs invalid branching (bracketing operators [and]) [6].

Ebner et al. use a fixed initial population where every individual has the single identity rule $f \rightarrow f$ [8].

In Hornby and Pollack's representation, they first generate a conditional predecessor by selecting a random parameter and a random constant to compare against. These are inserted into the formula $parameter > constant$ as the rule predecessor. Then, the successor string is generated by creating random blocks of two to three characters with parameters. [15]

2.2.4 Selection

The *Selection* step involves selecting individuals from the current generation of the population to be used in the next generation. The basis for the selection is the fitness of the individuals calculated by a fitness function. There are three main ways of calculating the fitness: evaluating at the properties of the interpreted L-system, evaluating how similar the interpreted L-system is to a target object, and human-based evaluation.

Various research evaluates various properties of the interpreted L-systems. Jacob first uses a simple method of rewarding L-systems with a majority of leaves between an inner cube and an outer cube [13]. This created "densely packed structures with broad branching" [13].

While this fitness function was not aimed particularly at evolving plants, Jacob later used a different fitness function aimed at "breeding artificial flowers" [13]. Here the fitness function rewards systems that spread out far in all directions and have as many blooms as possible.

Ashlock et al. use a similar measure to Jacob's bounding cube, but for 2D plants and a bounding area shaped more like a triangle. They calculate the fitness as "the number of pixels drawn inside the arena minus the number drawn outside". [10]

Mock's fitness function rewarded plants that where short but wide [6]. He also states that this was a simple fitness function compared to the real factors that determines a plants survivability, including "height, root depth, leaf size, attractive flowers, ability to withstand wind, proximity to water, etc." [6]

Positive phototropism (the movement of a plant based on stimulus by light) is

used by both Ochoa and Corchado et al. as part of the fitness function [7, 12]. They use a simplified version where taller plants are rewarded more. In reality, the positive phototropism direction could be different because the sunlight does not always arrive from directly above.

Bilateral symmetry is used by Ochoa as a measure of how balanced the weight of the plant is [7]. In the case of a 2D plant, this is done by finding the leftmost and rightmost points and calculating their proportion. The plant is rewarded more the closer the proportion is to 1.

Multiple researchers use light gathering ability as a measure of fitness. Ochoa calculates the total surface area of the leaves on the plant not shadowed by other leaves [7]. They assume that light arrives as vertical lines from the top. Ebner uses a similar method by using the OpenGL depth buffer to render the visible leaf area and count the pixels [9]. Corchado et al. also use a similar method, but instead of calculating the visible leaf area, they calculate the ratio of shadowed leaves to visible leaves and reward ratios closer to 0 [12]. Therefore, if there are no leaves at all, the plant would get a full score, which seems wrong as it can not gather any light.

Ochoa uses an additional metric that measures light gathering ability, but also seed dissemination ability. They assume that more branches improve these abilities, so they count the amount of branching points with more than one branch leaving it, and reward plants with a higher amount of these. [7]

Another metric used by multiple researchers is structural stability. Ochoa assumes that branching points with a high number of branches makes the plant unstable, so they punish plants based on the proportion of these branching points [7]. Corchado et al. use a similar method, but also considers branching points with few branches as unstable, because they are poorer at gathering light [12].

Ebner also considers this, but does so in a different way and calls it “structural complexity” [9]. They assign a cost of 1 to each branch segment, and a cost of 3 to each leaf. This cost is then multiplied with a factor that grows with the distance from the root. Thus, plants with long branches and many leaves are more structurally complex and are punished more (though this may be redeemed by their light gathering ability).

Beaumont and Stepney measure the similarity between the interpreted 2D L-system (plant) and a target 2D plant. They do this by looking at each drawn pixel in one image and calculating the distance to the nearest drawn pixel in the other image.

Ding et al. measure the similarity between an interpreted 3D L-system plant and a target image of a real tree. They combine the similarity of the outlines, topology and internal space into the fitness function. [29]

While Mock bred L-systems using a genetic algorithm, he also hand-bred plants using humans to select the “most interesting plant from each generation” [6].

McCormack used a similar method when they looked at aesthetic evolution of L-systems. They use a method called “interactive aesthetic evolution” where a normal GA process is used, but where humans select which individuals are to be used for the next population. They found that interactive evolution is still superior to autonomous evolution in terms of creativity. Interactive evolution can evolve plants into something that better match a human’s subjective criteria, while autonomous evolution only evolves plants into a similar style. [30]

2.2.5 Genetic Operators

After selection, the individual L-systems will undergo some genetic operations that modify or combine genetic material. The two most common operations are crossover and mutation, while there are some other operations used in special cases.

The crossover operation varies based on different genetic representations. For GP, where it is represented as a tree structure, Jacob swaps two expressions that have matching heads [13, 16].

For GA when the genotype is represented as a string, the operation varies based on what the string represents in the L-system. Mock and Corchado et al. exchange production substrings with equal number of push and pop bracket operators to avoid invalid syntax [6, 12]. Ochoa uses an approach inspired by GP by considering the successor string as a tree where nodes are defined by branches, and these nodes are exchanged [7]. Hornby and Pollack create a copy of one parent, and replace either a complete successor or substring of a successor with an equivalent part from the other parent [15]. Ebner et al. use two types of crossover operators: one-point crossover that exchanges rules and sub-tree crossover that exchanges sub-trees [8, 9]. The latter is similar to Ochoa’s method.

For genotypes encoded as arrays of real parameters, Ashlock et al. use a one-point crossover on said array [10]. For the genotype used by GE, which is encoded as integral numbers, Beaumont and Stepney also use one-point crossover [11].

The mutation operation also varies between different genotype representations. Unlike crossover, where, in most cases, there is only one operator, mutation often includes multiple different operators that mutate the genotype in various ways.

Jacob uses four different mutation operators for mutating GP trees. The first is to replace a sub-expression with a new generated matching sub-expression. The second is to delete sub-expressions. The third is to duplicate a sub-expression, creating a copy of it. The fourth and final is to permutate the parameters of an expression by shifting it left or right, or reversing it. For example, a sequence of

characters in a successor may be reversed (e.g. $ABC \rightarrow CBA$). [13, 16]

For string representations, there are multiple operators including replacement, addition, removal, swap and some specialized operators. Replacement involves replacing some part of the string with newly generated content. A valid sub-string may be replaced by a new valid sub-string [6, 12]. A symbol may be replaced by a new symbol [15, 8, 9, 12, 29], or a new valid sub-string [7]. The content of a branch may be replaced by a new valid sub-string [7].

New content can be added by an addition operator or existing content can be removed by a removal operator. For example, a symbol, branch or complete rule may be added or removed [8, 9], Hornby and Pollack also use the addition and removal operators, but on symbol blocks and symbol parameters [15].

In addition to replacing, adding and removing, symbols may also be swapped, simply by swapping their positions [8, 9]

Specialized operations are used for parametric L-systems to mutate the parameters of the symbols. Parameters may be added to or subtracted from, or if the parameter is an equation, the equation may be changed to a production. As parametric L-systems also may contain conditions, the equation of the condition may be mutated in a similar manner. [15]

Beaumont and Stepney use elitism where a certain amount of the most fit individuals move to the next generation without being mutated. They find indications that “elitism is required to ensure that good solutions are kept and built on, not lost or mutated in the next generation”, and therefore assign 20% of the population to be elites. [11]

McCormack and others mention that one needs to use measure to avoid invalidating bracket operators in a string. This may be done by excluding brackets from mutations, selecting sub-string within bracketed pairs, or repairing the brackets after mutation. [30]

3 Distribution-based Grammatical Evolution of L-system

3.1 Overview

To generate L-system plants that are aesthetically pleasing, varied and could be used to create offspring similar to its parents (**RQ0**), distribution-based grammatical evolution of L-systems (DGEL) was developed.

The DGEL name consists of two parts that are essential to solve the three sub research questions: *GE* and *Distribution-based*. It uses GE to evolve L-system plants because it allows the algorithm to use well-researched techniques for modifying or combining genes into new L-systems (**RQ1**). As part of the GE process, there is a fitness function that evaluates how aesthetically pleasing a plant is, thus pushing the evolution towards aesthetically pleasing plants (**RQ2**). It is distribution-based, meaning that different probability distributions for the L-system grammar can be used to generate varied plants (**RQ3**).

Figure 2 illustrates the modules, processes and models of the DGEL algorithm. It consists of two modules: *Evaluation* and *Generation*. Evaluation evaluates how good an L-system plant is, and Generation uses this information to generate good L-system plants. The end result is a plant 3D model that is interpreted from an L-system. The L-system is generated by a chromosome using a grammar distribution for a grammar description. These three models: grammar description, grammar distribution and chromosome, together define the L-system. All L-systems generated using the same grammar description, grammar distribution and chromosome will result in the exact same L-system, which also will be interpreted into the exact same plant 3D model. GE is used to generate a good chromosome based on the grammar description, grammar distribution and a fitness function. The grammar distribution is generated by SA based on the grammar description and a fitness function. The fitness function models how aesthetically pleasing a plant is by assigning it a score in range $[0, 1]$, where 0 is the worst, and 1 is the best.

3.2 Representing an L-system with Grammar

As described, the L-systems in DGEL are represented by a chromosome, a grammar description and a grammar distribution. If we assume a uniform grammar distribution, only a chromosome and a grammar description is required.

The grammar description, represented in Augmented BNF (ABNF) [31], de-

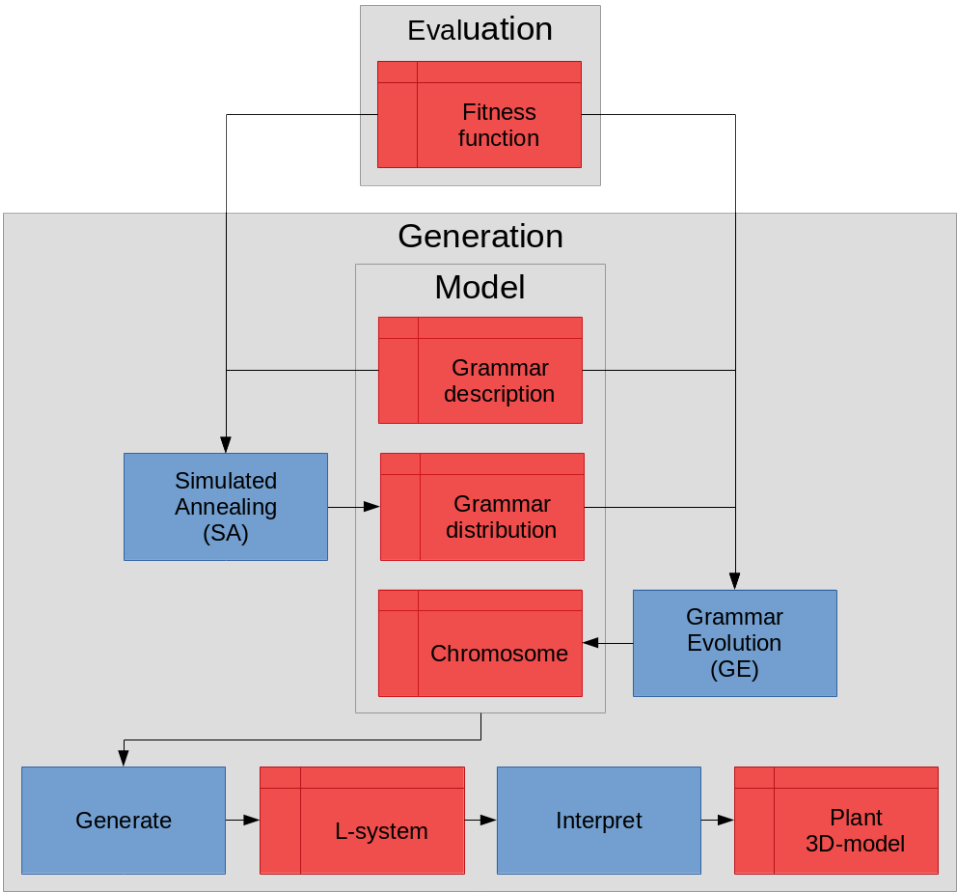


Figure 2: DGEL components and flow. Blue boxes represent processes and red boxes represent models. Grey boxes are logical collections of the components.

scribes the syntax of the L-system. For example, it may describe that an L-system consists of a number of rules, each with a predecessor and successor, where a predecessor contains one character. The chromosome describes the choices to be made in the grammar description, for example how many rules there are, or which character the predecessor contains. These choices are what creates the variations in the L-systems. If there were no choices, there would only exist one single L-system.

This method of representing the L-systems was based on Beaumont and Stepney's work on applying GE to plant L-systems [11], in addition to Ortega et al.'s work on applying it to fractal L-systems [27], and Ryan et al.'s introduction of GE [28]. Though instead of using traditional BNF, ABNF is used because of its support for repetitions with minimum and maximum, grouping and value ranges, and the fact that it has a simpler syntax [31].

While Ortega et al. and Beaumont and Stepney make assumptions about the grammar to significantly simplify it and reduce the search space [27, 11], the DGEL method uses a grammar that covers the largest search space reasonably possible. To make the search space reasonable, there still has to be some limits. The number of productions, string length, and number of characters for a variable were limited to a maximum of 20. A higher number did not seem reasonable as no L-systems in the literature reviewed used larger numbers, and the search times for higher numbers started to become unreasonably long. Additionally, to limit the complexity of L-systems, complex features such as leaves, flowers, branch segment width and more were not included. Thus the grammar used can only produce the branches of a plant. This is the most essential part of a plant, which can be expanded later by modifying only the grammar description. Listing 1 shows the actual grammar description used during the development and testing of the algorithm, while Listing 2 shows a grammar description that supports leaves and varying branch segment width.

To be able to use different grammar distributions, the chromosome representation is a different from that of Ryan et al. where each gene is an unsigned 8-bit integer and the alternative to be selected is the alternative indexed by the gene value modulo the number of alternatives [28]. Each gene is still represented by an unsigned integer, but it is 32-bit instead of 8-bit, and the interpretation of the integer is different.

A gene in the DGEL chromosome can be viewed as an index into the value range of the 32-bit integer. This value range, from 0 to 2^{32}, can be viewed as a line with segments representing each alternative in a rule in the grammar. In the case of a uniform distribution the segments would be equal in length, while for a different distribution the lengths will differ. The value of each gene in the chromosome is randomly picked from a uniform distribution, thus making the actual distribution

```
lsystem = axiom productions
axiom = string
productions = 1*20production
production = predecessor successor
predecessor = variable
successor = string
string = 1*20(symbol / stack)
stack = "[" string "]"
symbol = variable / operation
variable = %x41-55
operation = "+" / "-" / "^" / "&" / ">" / "<"
```

Listing 1: ABNF grammar description used in DGEL

```
lsystem = axiom productions
axiom = string
productions = 1*20production
production = predecessor successor
predecessor = variable
successor = string
string = 1*20(symbol / stack / leaf)
symbol = variable / operation
variable = %x41-55
operation = "+" / "-" / "^" / "&" / ">" / "<" / "!"
stack = "[" string "]"
leaf = "['{+f-f-f+|+f-f}]"
```

Listing 2: ABNF grammar description supporting leaves and varying branch segment width

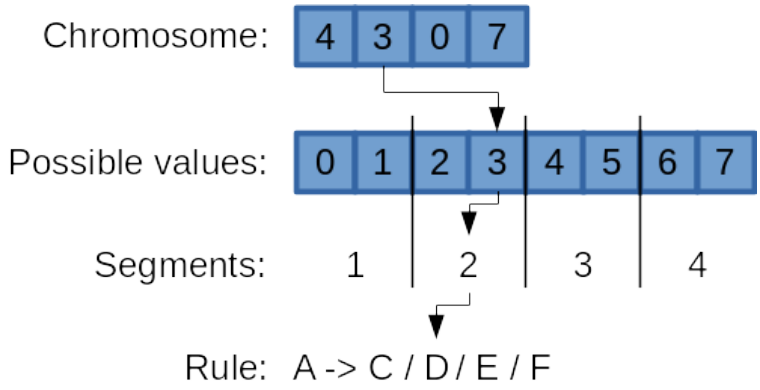


Figure 3: Example grammar alternative selection from gene. In the figure, the gene is a 3-bit integer, and therefore has 8 possible values.

dependent on the grammar distribution.

Figure 3 illustrates an example of this process. A choice is to be made between C, D, E and F in the rule $A \rightarrow C/D/E/F$. A chromosome of length 4 containing 4, 3, 0 and 7 is used, and the next gene to be used in the chromosome is the second position, i.e. the value 3. Each gene is a 3-bit unsigned integer, therefore having 8 possible values in range $[0, 8)$. Since there are four alternatives, and a uniform distribution is used, the 3-bit value range is split into four segments of 2 values. The gene value 3 maps into the second segment, which maps to the second alternative in the rule, i.e. D. This process would then continue with the next gene, 0, on the D rule.

Obviously, a 3-bit integer can not always be split uniformly. For example, if there are three alternatives, the closest approximation would be to have two segments cover three values and one cover two values, making one alternative have 12.5% less chance of being selected. An additional problem is that when the number of alternatives is higher than the available values, some alternatives are forced to have 0% chance of being selected. These issues can be mitigated by increasing the range of the integer, in this case to a 32-bit unsigned integer (used by DGEL). To prove this, consider the previous example only with a 32-bit integer instead. Splitting it into three segments would yield two segments of size 1,431,655,765 and one of size 1,431,655,766 (1 in difference), and all of them would have a 33.33% of being selected (rounded to two decimal places). Even without rounding the value, there is only a 2.33×10^{-8}% difference between them.

Character	Operation
F	Draw line forward
+	Yaw left
-	Yaw right
^	Pitch up
&	Pitch down
<	Roll left
>	Roll right
|	Yaw 180°
[	Push state
]	Pop state
{	Begin surface
}	End surface
!	Decrease width
‘	Next color
f	Draw line forward

Table 1: Turtle interpretation of L-system characters. `f` does the same as `F`, but is not allowed as a predecessor int the grammar so that it can not recurse.

3.3 Interpreting L-systems Into Plant 3D Models

As seen in Figure 2, after an L-system has been generated, it needs to be interpreted into a 3D-model such that people may see the plant. Turtle interpretation is used to do this because it is a popular method for 2D and 3D structure generation based on L-systems [1]. A 3D version of the turtle interpretation is used, where the turtle can draw lines with `F`, and rotate around three axes using `-` and `+` for yaw, `^` and `&` for pitch, and `<` and `>` for roll. The bracket operators, `[` and `]`, are used to create branch bases. Table 1 shows a complete overview of the characters and their operations, including extra operators used for more control and features like leaves (e.g. Listing 2).

The actual 3D model created from this interpretation is kept as simple as possible while still trying to look somewhat natural. The branch segments (created by the `F` character) are brown stretched cubes. Leaves are green flat surfaces. An example can bee seen in Figure 4.

As a discrete L-system is used, the turtle interpretation requires some parameters that control the drawing. While some evolutionary L-system techniques evolve the turtle interpretation parameters as well as the L-systems, to keep the process simple, DGEL does not. Additionally, if the discrete L-system is swapped with a

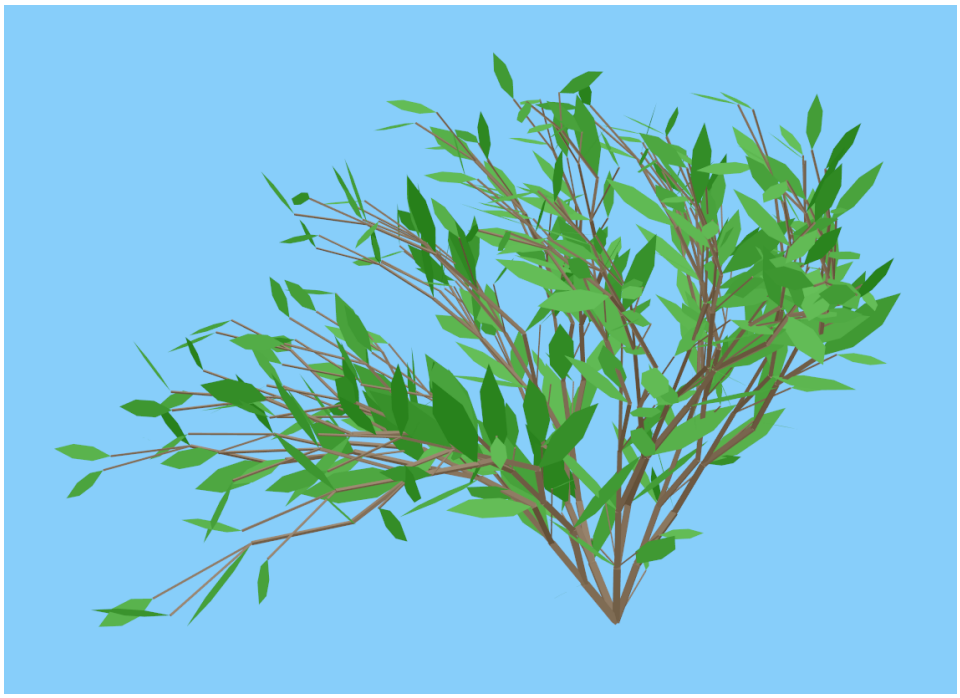


Figure 4: Example 3D plant model generated by DGEL

Parameter	Value
Width	0.05
Angle	22.5°
Iterations	5
Step	0.2

Table 2: Turtle interpretation parameters

parametric L-system, these parameters are no longer required. The parameters used here were based on the 3D bush and 3D plant presented by Prusinkiewicz and Lindenmayer (Figures 1.25 and 1.26) [1]. They can be seen in Table 2.

As described earlier, leaves and branches with varying widths are not included. Without these, the plants may look “dead”, and so can not be considered very aesthetically pleasing. To work around this, widths of branch segments were heuristically set and leaves were heuristically placed. These heuristics were based on observations of real plants and L-system based 3D models such as the aforementioned models presented by Prusinkiewicz and Lindenmayer (Figures 1.25 and 1.26) [1].

The branch segment widths are determined by their distance from the branch top. From a branch top, the segment width will increase by a fixed number towards the first branch base. In a branch base, the width will be set to the largest of the child branch segments. Then, the width will continue to increase towards the tree root. Thus the plant will have natural-looking branches that are larger closer to the root. Bigger plants will naturally have thicker branches than smaller plants.

Leaves are placed from the branch tops towards the root on each branch segment in increasing sizes, until they become too big. Thus, the leaves on the branch tops are the smallest, and they get bigger towards the root. To prevent thick branch segments, e.g. the trunk of a tree, to have leaves directly placed on them, the leaves will no longer be placed when they reach a certain size. The leaves are always pitched upwards, with a random noise applied to the pitch to look more natural. They have a random rotation around the branch segment they are placed on.

This interpretation of the L-system can be swapped out with a different interpretation, while still using the DGEL algorithm. For example a more realistic interpretation with smoothed branches or branches bent by gravity could be used. Thus the DGEL algorithm is not dependent on this particular interpretation, but the results described in Chapter 5 will be.

3.4 Using a Grammar Distribution to Limit the Search Space

In DGEL, grammar distributions are used for three reasons: to guide the GE to more efficiently find good plants, to guide GE to search in a space with certain types of

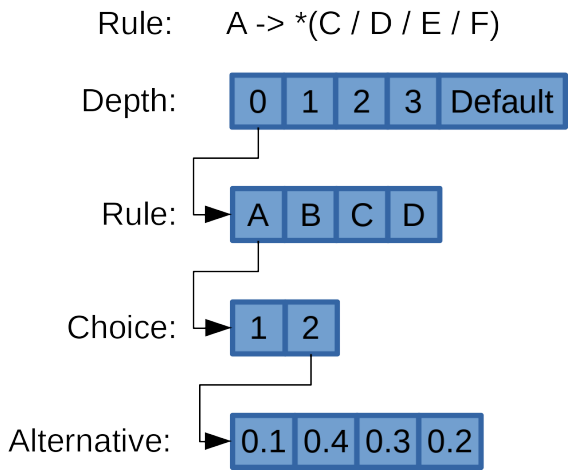


Figure 5: Example grammar distribution applied to the second choice of rule A at depth 0.

plants, and to prevent infinite recursion.

The distribution is a set of weights per choice per rule per depth, as illustrated in Figure 5. The depth is the current branching depth of the L-system, controlled by the bracket operators, and defined as the `stack` rule in the grammar in Listing 1.

The example figure shows a case where a choice is to be made between alternatives C, D, E and F in rule $A \rightarrow *(C/D/E/F)$. There are two different choices in the rule: selecting the number of repetitions (indicated by the asterisk character, $*$), and selecting the C/D/E/F alternative. In the example, the depth is 0 (no branching), the rule is A and the choice is the second. This maps to the weight array: $[0.1, 0.4, 0.3, 0.2]$ which should be applied to the C/D/E/F choice. Thus, C has a 10% chance of being selected, D has 40%, E has 30%, and F has 20%. These weights are then used to define the segments in the gene, as described in Section 3.2.

If no weights are found in the distribution for a certain choice at a certain depth, the weights in the "default" depth will be used, if they exist. If not, the weights will be uniform. This can be used to prevent infinite recursion with the `stack` rule. For example, the distribution for the grammar in Listing 1 may specify the weights $[0.5, 0.5]$ for the second choice (`symbol / stack`) in the `string` rule at depth 0, 1 and 2, such that there is an equal chance to produce a string and a stack (branch base). To prevent any deeper branch bases, the weights $[1.0, 0.0]$ may be specified in the default depth, causing it to not produce a stack at depth 3 or deeper.

By using a specialized distribution, rather than a uniform distribution, the gen-

erator may be targeted at specialized L-systems, or simply have a higher average fitness score for the generated plants. For example, the distribution may focus more on producing the `F` character which is used for drawing the branch segments. Or it may be focused on the `stack` rule that produces branch bases. This may in turn increase the rate of produced plants that have long or many branches respectively.

In order to find specialized distributions that more efficiently generate good plants or generate different types of plants, the weights in the distribution need to be optimized. As the distribution maps from depth, to rule, to choice, to alternative (weight), it is a four-dimensional parameter space. DGEL uses SA to perform the optimization, but other multi-dimensional parameter optimizers could just as well be used.

Specifically, DGEL implements the basic SA approach [32], as described by Onbaşoğlu and Özdamar [33]. The only difference is that the initial solution is not initialized randomly, but initialized from a pre-optimized solution, so that it has a better starting point. In the DGEL SA, the *solution* is the distribution itself, filled with weights for each of the alternatives in each of the choices in each of the rules in a limited set of depths.

A neighbor solution is generated in the same manner as Onbaşoğlu and Özdamar describes [33]: selecting one single random weight and either increasing or decreasing the weight randomly, bounded to the range $[0.0, 1.0]$ in the case of DGEL. A key difference is that since these are *weights*, the remaining weights in the same set of weights have to be modified such that the sum of all of the weights equal 1. If the sum of the remaining weights is greater than 0, they can be scaled by a normalization factor, calculated by $\frac{(1-w)}{(1-w_0)}$, where w is the new value of the modified weight, and w_0 is the old value of the modified weight. If the sum of the remaining weights is exactly 0, scaling the weights will not work, as they would remain 0. The alternative solution is to divide the "remaining weight" over the remaining weights. This is done by setting the remaining weights to $\frac{(1-w)}{(n-1)}$, where w is the new value of the modified weight, and n is the number of weights.

Accepting a neighbor solution also follows the same approach as Onbaşoğlu and Özdamar describes [33]. To do this, an $f(x)$, where x is a grammar distribution, has to be defined. The performance of a grammar distribution can not directly be measured, but it can be estimated by measuring the average performance of the L-systems it generates. As the performance of the L-systems generated may vary by a big degree, the average performance, $mean$, is only accepted after a specified amount of L-systems, n, have been generated, and the standard error of the performance, se, is below a threshold value, t (Algorithm 1).

Algorithm 1 Distribution evaluation

```
function EVALUATE_DIST(dist, n, t)
    samples ← EMPTY_LIST
    mean ← UNDEFINED
    se ← INFINITE
    while se ≥ t do
        new_samples ← GENERATE(n,dist)
        samples ← CONCAT(samples,new_samples)
        mean ← MEAN(samples)
        se ← STANDARD_ERROR(samples)
    end while
    return mean
end function
```

3.5 Using Grammatical Evolution to Search the Space

The main component of DGEL is the GE of the L-systems, which is used to search the parameter space for the best L-system. The GE process takes a grammar description and a grammar distribution as the input, and produces a single L-system chromosome as the output.

It follows the process as described by Ryan et al. [28], but with some modifications to the representation as described in Section 3.2. Ryan et al. do not describe the selection strategy they used, while Beaumont and Stepney use a simple selection strategy where a percentage of the best individuals are selected as parents [11]. Since GE is a specialization of GA, GE can use the same selection strategies as GA. Thus, tournament selection was selected for DGEL based on Blickle and Thiele's structured comparison of selection strategies [34], and Razali and Geraghty's comparison of selection strategies applied to the traveling salesman problem [35]. Both tournament selection and ranking selection are recommended as good options, but tournament selection performs faster [34]. Therefore, because it is assumed that DGEL may be used in real-time applications, tournament selection is preferred.

In order for the choice of tournament selection based on Blickle and Thiele's comparison to be valid, the GE algorithm is implemented as GA is described by Blickle and Thiele [34]. The only difference from the GA algorithm is the addition of the duplication and prune operators that were specifically added to GE [28].

3.6 Using a Fitness Function to Evaluate Plants

A crucial component of both the SA and GE process is the fitness function. The fitness function is used by SA to evaluate the grammar distributions, and by GE to evaluate individual L-systems. Its function is to evaluate how "good" the L-system

Metric	Description	Equation	Weight
Nothing	Punish plants that have no branches (can not be seen)	3.1	1
Closeness	Punish plants with child branch segments very close to each other	3.2	1
Drop	Punish plants based on how much the plant grows downwards	3.3	1
Balance	Reward plants that have their center of gravity closer to their root	3.4	1
Branching	Reward plants with a balanced number of child branches	3.5	1
Foliage	Reward plants with more leaves	3.6	1.5
Length	Reward longer plants	3.7	1
Curvature	Reward plants that have somewhat curving branches	3.8	0.4

Table 3: Fitness metrics

plants are, or more specifically how "aesthetically pleasing" they are. The literature on evolutionary L-systems generally aim the fitness function at evaluating how realistic the plants are by using metrics such as its light gathering ability, positive phototropism and structural stability.

It is assumed that a baseline for aesthetically pleasing plants is that they look at least somewhat realistic, otherwise they would not look like plants at all. Thus, the metrics used for the fitness function in DGEL uses metrics from the literature, or metrics inspired from the literature, where the metrics usually are targeted towards realistic plants. Additionally, the metrics were improved upon by generating a plants and looking for bad features in well-scored plants. Following Ochoa [7], each metric is weighted so that their importance may be tuned. Table 3 lists all of the metrics and their weight.

The fitness metrics can be either rewarding, punishing, or both. Rewarding metrics give a score of $[0, 1]$, while punishing metrics score them $[-1, 0]$ and metrics that are both score them $[-1, 1]$. Metrics that use the full range are considered punishing if the score is negative and rewarding if not. Having these different ranges makes it possible to intuitively understand if the plant is good, bad or OK in terms of a metric. The final fitness score is an summation of the rewarding and punishing scores, transformed to range $[0, 1]$: $\frac{\mathrm{reward}+\mathrm{punishment}+1}{2}$.

Nothing is the simplest and most basic metric. It punishes plants that have no branch segments. This may happen if for example the L-system contains no drawing commands (the character `F`). Equation 3.1 demonstrates this by giving it a score of

-1 if it does not have any branch segments. In this case no other metrics will make any difference. The $\mathtt{branches}$ function returns the number of branch segments.

Because of the recursive property of L-systems and the flexible grammar used in DGEL, the expanded string from running all the iterations may become very large. For example, if the axiom contains the letter F, and the production F produces 20 F, with 5 iterations, the expanded string would have length 20^5. Additionally, the interpreted 3D model would consist of $20^5 + 1$ points, which is too much for the simple 3D renderer used. Therefore the number of points in the interpreted 3D model is limited to 10,000 (skeleton limit). If the F was instead a rotation operator, 20^5 matrix multiplications would have to be performed, and the skeleton limit would not be breached. Thus the number of instructions (i.e. characters in expanded string) is limited to a maximum of 50,000 (instruction limit). Both limits were experimentally found to be balanced in time and space, while still allowing complex structures. If they are breached the L-system will become *nothing* and thus be scored zero in the final fitness score.

$$\mathtt{nothing}(x) = \begin{cases} 0, & \text{if } \mathtt{branches}(x) > 0 \\ -1, & \text{otherwise} \end{cases} \tag{3.1}$$

Closeness is another punishing metric that punishes plants that have branch bases with child segments too close to each other. This metric helps avoid plants that have segments clipping into each other. This clipping may look bad, and if they are completely inside each other the complexity of the 3D model increases, which decreases the rendering performance without improving visuals. The dot product between the direction of two child segments is used for this metric. The smaller the angle between the two direction vectors is, the larger the dot product is, becoming 1 when they are parallel. If the largest dot product between segment directions in a branch base is below a threshold, t, there is no punishment. Otherwise, the plant is punished by a factor depending on how much above the threshold the dot product is. This factor is interpolated between 0 and 1 linearly from the threshold to 1. Equation 3.2 shows how the closeness is calculated. The $\mathtt{closest}$ function finds the closest dot product between child segments in a branch base. The threshold was experimentally set to 0.9 to punish plants that have segments that look like they clip into each other.

$$\mathtt{closeness}(x) = \begin{cases} 0, & \text{if } \mathtt{closest}(x) < t \\ -\frac{c-t}{(1-t)}, & \text{otherwise} \end{cases} \tag{3.2}$$

Drop is a third punishing metric that punishes plants that grow downwards. It finds the lowest point on the plant, and punishes the plant if it is below 0. Equation 3.3 demonstrates this. The $\mathtt{lowestpoint}$ function returns the smallest value of

the y-component of all points on the plant. This value is then clamped in the range -1 and 1 and interpolated using a sine function to quickly increase the punishment. Thus plants where the lowest point is 0 or above will not be punished, while on plants where it is -1 or below will be punished by -1.

$$drop(x) = sin(clamp(lowestpoint(x), -1, 0) * \frac{\pi}{2}) \tag{3.3}$$

Balance is a measure of how well balanced the weight of the plant is. It is inspired by the Bilateral Symmetry measure by Ochoa where 2D plants that reach equally far both left and right are rewarded the most [7]. A different approach has to be taken in 3D where there are not only two horizontal directions, but rather an infinite number. The approach taken is to estimate where the center of gravity of the plant is, calculate the horizontal distance, $centerdistance$, to it, and compare it to how far the plant reaches, $centerspread$, in the horizontal direction of the center of gravity. This is done by Equation 3.4. In the worst case, if both $centerdistance$ and $centerspread$ are the same, the plant will be punished by -1. In the best case, if the $centerdistance$ is 0, i.e. the gravitational center is in the center of the plant, the plant will be rewarded by 1. Since $centerdistance$ generally will increase with increasing $centerspread$, the plant will be more punished the more it spreads out in one direction. Though by having most of its branch segments close to the root, it may mitigate the punishment. Thus the fraction $\frac{centerdistance}{centerspread}$ is used.

$$balance(x) = (0.5 - \frac{centerdistance(x)}{centerspread(x)}) * 2 \tag{3.4}$$

Branching measures how complex the branching of the plant is. It is inspired by the Structural Stability [7], Plant Structural Stability [12] and Proportion of Branch Bases [7] measures. These measures assume that the plant becomes too unstable to survive if it has too many child segments growing from a branch base. Corchado et al. additionally assume that plants with too few branches will have worse light gathering ability. Thus there will be a sweet spot in the branching proportion which is rewarded the most. If the proportion is too low or too high, the plant will be punished. This is shown in Equation 3.5 where if the branching proportion is below 1.2 (1.2 child segments from a branch base on average), or above 5, it will be punished in range $[-1, 0]$. While between 1.2 and 5, it will be rewarded in range $[0, 1]$. The sweet spot is 2, where it will be rewarded by 1. $icos$ interpolates the value from the first parameter, a, to the second parameter, b, using a cosine function. The third parameter, t, is in range $[0, 1]$ and controls where between a and b the interpolation is. This creates a smooth curve between

the limits.

$$\mathit{branching}(x) = \begin{cases} -1, & b < 1 \\ \mathit{icos}(-1, 0, \frac{b-1.0}{0.2}), & b < 1.2 \\ \mathit{icos}(0, 1, \frac{b-1.2}{0.8}), & b < 2 \\ 1, & b < 3 \\ \mathit{icos}(1, -1, \frac{b-3}{5.8}), & b < 7 \\ -1, & \text{otherwise} \end{cases}, \quad \text{where } b = \frac{\mathit{branches}(x)}{\mathit{branchings}(x)} \tag{3.5}$$

Foliage measures how many leaves a plant has. It is the main metric for measuring the light gathering ability of the plant, and is inspired by Ochoa [7], Ebner [9], and Corchado et al.'s [12] light gathering ability metrics. The main point is that more leaves exposed to sunlight means that the plant has a better ability to survive. For simplicity, the metric used in DGEL is based on Corchado et al.'s metric which simply rewards plants with more leaves [12]. It also counts leaves that are shadowed by other leaves, even though in reality these would gather only minimal amounts of light. This selection of method was due to time constraints, and a more complex method like Ebner's would be better. The metric used starts with a reward of 0 for 0 leaves and is asymptotic towards 1 when the leaf count, found by leaves, increases towards ∞. Equation 3.6 demonstrates this. The s constant controls how fast it should increase towards 1.

$$\mathit{foliage}(x) = \frac{\mathit{leaves}(x) * s}{1 + \mathit{leaves}(x) * s}, \text{where } s = 0.1 \tag{3.6}$$

Length rewards plants that are longer. It is inspired by Ochoa and Corchado et al.'s Positive Phototropism metrics where the assumption is that plants grow towards the sunlight and therefore taller plants are better at surviving. While Positive Phototropism only cares about the height of the plant, Length cares about how long the plant is, not considering if it grows upwards or not. This change is made on the assumption that plants can gather more light both by growing upwards (cover more vertical area) and sideways (cover more horizontal area). Additionally it was found during an early experiment that short plants were generally displeasing. Like the Foliage metric, the Length metric is asymptotic towards 1 as the plant length increases towards ∞. This is shown in Equation 3.7. The length of the plant is found using $\mathit{longest}$, which finds the longest path in the branches of the plant from root to leaf and returns the number of branch segments in it. As with the Foliage metric,

the s constant controls how fast it should increase towards 1.

$$\mathit{length}(x) = \frac{\mathit{longest}(x) * s}{1 + \mathit{length}(x) * s}, \text{where } s = 0.5 \tag{3.7}$$

Curvature is a metric that rewards plants with more curved branches. It was developed from the results of the first experiment where it was found that plants that looked "static" or "straight" were not as pleasing as plants that looked more "dynamic" or "curvy". As Equation 3.8 demonstrates, the plant is rewarded more if the branch segments are oriented with a small angle different from their parent. If the angle is too sharp, the plant is rewarded less, as the plant may look like "doodles" with such angles. The $\mathit{minangles}$ function returns a list of the smallest angle from an internode segment to its parent segments. Thus, if an internode segment only has one child (i.e. it does not branch), the smallest angle will be the angle between it and that one child. Otherwise, if an internode segment has multiple children (i.e. it is a branch base), all of the angles between it and the children are calculated and the smallest will be used. This is based on the assumption that the child segment with the smallest angle is the straight segment following a branch base. avg calculates the average of all of the angles returned by $\mathit{minangles}$. Like in Equation 3.5, icos is used to create a smooth interpolation between the limits min, opt and max. A sweet spot of $\mathit{opt} = 0.24711092$, which is rewarded by 1, was found to provide a good curvature in the plant. A limit of $\mathit{max} = \frac{\pi}{4}$ (45°) was intuitively chosen as a point where the curvature is no longer rewarding.

$$\mathit{curvature}(x) = \begin{cases} \mathit{icos}(0, 1, \frac{a - \mathit{min}}{\mathit{opt} - \mathit{min}}), & a >= \mathit{min} \text{ and } a < \mathit{opt} \\ \mathit{icos}(1, 0, \frac{a - \mathit{opt}}{\mathit{max} - \mathit{opt}}), & a >= \mathit{opt} \text{ and } a < \mathit{max} \\ \mathit{min}, & \text{otherwise} \end{cases},$$
$$\text{where } a = \mathit{avg}(\mathit{minangles}(x)), \tag{3.8}$$
$$\mathit{min} = 0,$$
$$\mathit{opt} = 0.24711092,$$
$$\mathit{max} = \frac{\pi}{4}$$

4 Implementation of DGEL

4.1 Programming Language

DGEL is implemented in Rust, a systems programming language [36]. The choice of Rust was based on multiple factors. Performance is an important factor as complex processes requiring many iterations are to be run (GE and SA). Additionally abstractions are desired to improve ease of use and reasoning of the code. Rust fares well in this part because it is a compiled language with zero-cost abstractions, making its performance competitive with that of C and C++ [37].

Another benefit of using Rust over for example C++ is its focus on safety. Rust, by the language design itself, does not allow unsafe operations, thus guaranteeing in most cases that there will be no segfaults or data races [38, 36, 39]. The guarantee about data races is particularly useful since it makes the writing of parallel code easier and safer, which the DGEL processes can benefit heavily of. Finally, as Rust uses LLVM[1] to compile to machine code, it has good cross-platform support.

Although Rust is fairly new[2], it already has a large repository of libraries, including 12,767 published crates (Rust's name for an external library) as of December 2017[3]. This has aided the implementation of DGEL.

4.2 Overview

The DGEL implementation is part of a larger application designed to experiment with L-systems. It implements the modules `dgel` and `fitness` within the program `lsystem`. `dgel` contains the implementation of the DGEL algorithm itself, and functionality to experiment and test DGEL. `fitness` contains the fitness function with all of the metrics implemented. Four independent crates that are used by `lsystem` for various purposes are additionally implemented: `abnf`, `lsys`, `lsys_kiss3d`, and `yobun`. The source code for the complete application is archived on Zenodo [40].

`yobun` is the simplest crate and consists of utility functions of various sorts, including interpolation functions, filesystem functions, parsing functions and more.

`abnf` implements parts of the ABNF syntax, file parsing and expansion of rules, which is a central part of `dgel`. The `parse` module within it implements the pars-

[1] The LLVM compiler infrastructure: `http://llvm.org/`

[2] First stable Rust version released in May 2015: `https://blog.rust-lang.org/2015/05/15/Rust-1.0.html`

[3] crates.io, The Rust community's crate registry: `https://crates.io/`

Language	Files	Lines	Code	Comments	Blanks
Rust	22	11879	9955	296	1628
TOML	5	88	72	5	11
YAML	2	84	84	0	0
Markdown	1	13	13	0	0
Total	30	12064	10124	301	1639

Table 4: Code count of whole application

ing of ABNF files using the `nom` create[4], which enables the composition of smaller parsers into larger ones. Each of the parsers are tested thoroughly using Rust's built-in unit testing feature. The parser parses the file into the structures defined in `syntax`, which represents the concepts in ABNF, such as alternatives and repetitions, and can be created or modified programmatically. The `core` module contains some of the ABNF core rules as defined in the RFC [31]. Finally, the `expand` module can expand the ABNF grammar from a rule into a full string using a specified `SelectionStrategy`. This is an abstract trait that can be implemented by the user of `abnf` to choose how to select between choices in the grammar. `dgel` uses this to implement the `ChromosomeStrategy`, `WeightedChromosomeStrategy` and `WeightedChromosomeStrategyStats`. The first is the implementation of the original GE chromosome, the second is DGEL's version with a grammar distribution, and the last is a strategy that gathers stats about which choices were made.

`lsys` implements L-system functionality, including D0L-systems (`ol`), IL-systems (`il`), parametric L-systems (`param`) and the expansion of them. It generalizes features of L-systems in the `common` module so that different L-systems have a common interface and can be easily swapped out. For example, `Instruction` represents a generic L-system instruction with parameters, `Skeleton` represents the structure of an interpreted L-system, and `Rewriter` has an interface to expand the L-system into a single string. In the case of `dgel`, only D0L-systems are currently used, and the others are there for other experimental reasons.

Finally, `lsys_kiss3d` is the L-system interpreter implemented using the Kiss3d crate for rendering[5]. It supports all of the instructions in Table 1, and has an alternative interpretation using the heuristics described in Section 3.3.

To give an idea of the scope of the program and its source code, the code count is shown in Table 4 (counted by Tokei[6]). The whole program consists of around 12,000 lines of code (including blanks). In addition to Rust files, TOML files de-

[4]nom crate: `https://crates.io/crates/nom`
[5]Kiss3d: `http://kiss3d.org/`
[6]Tokei code counting program: `https://github.com/Aaronepower/tokei/tree/v6.1.2`

Crate	Files	Lines	Code	Comments	Blanks
lsystem	10	7502	6320	146	1036
lsys	5	1889	1540	60	289
abnf	5	1356	1169	27	160
lsys_kiss3d	1	527	441	6	80
yobun	1	605	485	57	63

Table 5: Code count for each implemented crate

Command	Description
animated	Run animated visualization of plant growth
dgel	Run random plant generation using GE
generated	Run random generation of plant
measure	Measure the fitness of a model
static	Run visualization of static plant

Table 6: Commands available to the application

scribe the five crates implemented, YAML files describe L-system grammars, distributions and settings, and Markdown files are informative. In total there are 445 revisions of the code (managed by Git[7]).

As seen in Table 5, most of the code is in the main crate (`lsystem`), while substantial amounts are also in `lsys` and `abnf`. This is because the main crate includes both the GE process, the SA process and all of the command-line tools. Parts from both GE and SA could potentially be moved out into separate crates if they are generalized.

There are multiple command-line tools available in the application. Table 6 shows all top-level tools, Table 7 shows tools specific for the `dgel` command and Table 8 shows tools available for the `dgel ge` command. Note that the command descriptions are extracted from the help command and may be outdated. To run the complete DGEL process, the distribution first as to be generated using `dgel learning` (SA), then the GE process can be run using `dgel ge run -d <dist>`, where `<dist>` is the distribution generated by the previous step. Finally, `dgel visualized` can be run to visualize the L-system model that GE saved to the filesystem in a YAML file. Other commands are tools that aided the research in benchmarking code, generating statistics, sampling the space, recording videos of the plants for the survey in Section 5.2.2, and measuring L-systems.

[7] Git version control system (VCS): `https://git-scm.com/`

Command	Description
abnf	Print the parsed ABNF structure
bench	Run benchmarks
dist-csv-to-bin	Convert a CSV distribution file to a bincode file
dump-default-dist	Dump default distribution to files
ge	Run grammatical evolution
inferred	Run program that infers the genes of an L-system
learning	Run learning program until you type 'quit'
random	Randomly generate plant based on random genes and ABNF
record-video	Record a video of a plant model
sample-weight-space	Sample a weight space and write points to file
sampling	Run random sampling program until you type 'quit'
sampling-dist	Take samples from directory and output a distribution CSV file
sort-models	Sort stored models based on score
stats	Generate samples and dump stats to be analyzed
visualized	Generate plants based on a predefined distribution

Table 7: Commands available to the `dgel` command (`learing` is the command that runs the SA process and should be renamed to reflect this)

Command	Description
duplication-sampling	Find the best GE duplication rate
recombination-sampling	Find the best GE crossover and mutation rates
run	Run GE
size-sampling	Find the best GE population size and number of generations
tournament-sampling	Find the best GE tournament size

Table 8: Commands available to the `dgel ge` command

4.3 L-system expansion

The expansion of the L-systems is implemented as an iterator over the expanded instructions such that it can stop the expansion prematurely if it reaches the instruction or skeleton limit. To accomplish this, the iterator keeps a stack of the instructions that should be expanded next and the rewrite iteration it was expanded from. In each iteration of the iterator, it recursively rewrites the current instruction while storing production successors in the stack until it reaches the max number of L-system iterations, where it returns the current instruction.

4.4 Parallelized GE

The GE implementation is a modified version of the RsGenetic crate[8]. It was modified to match the GA algorithm as specified by Blickle and Thiele [34], in addition to supporting custom genetic operators, such that the duplication and pruning operators used by GE can be used.

The genetic operators use the Rayon crate[9] to parallelize all of the operators with parallel iterators. Thus the operators modify individuals from the population in parallel, allowing for much higher throughput on multi-core CPUs. It is an easy to use method that only involves including the crate and changing `iter()` to `par_iter()`, while allowing for a significant increase in performance. The tournament selection is parallelized in a similar manner, doing all the selections of new individuals in parallel.

4.5 Parallelized SA

The SA is parallelized in terms of the grammar distribution measurement as this is the most costly part; it has to generated several individual L-systems and evaluate their fitness before finding their average. This part is also perfect for parallelization because each L-system can be generated and evaluated completely independently, making it an embarrassingly parallel problem. The specific function that is parallelized is `GENERATE` in Algorithm 1. This function generates and evaluates n L-systems in parallel.

While it could be parallelized with Rayon, it instead uses the `futures_cpupool`[10] crate such that it may add other operations to the pool of parallel tasks. In this case it puts the saving of the SA progress into the thread pool so that it does not block while waiting for I/O.

[8] RsGenetic: `https://github.com/m-decoster/RsGenetic`
[9] Rayon: `https://github.com/rayon-rs/rayon`
[10] futures_cpupool: `https://crates.io/crates/futures-cpupool`

5 Evaluation of DGEL

DGEL consists of two main modules: *generation* and *evaluation*. The evaluation module consists of the fitness component which evaluates how good the L-system plants are. The remaining components are part of the generation component which generates plants, and uses the evaluation module to generate *good* plants. To answer if DGEL solves the research questions, both modules have to be evaluated. The generation module is evaluated in a technical manner to determine if it performs well or not, while the evaluation module is evaluated using humans as to answer if the fitness function properly rates plants as aesthetically pleasing or not.

5.1 Evaluation of the Generation Module

5.1.1 Method

As explained in Section 3.1 and shown in Figure 2, there are two main processes involved in generating the model for the L-system plants: *SA* and *GE*. The remaining steps are dependent on the model and do not create variations on the same model. Thus GE and SA are the two processes of interests to evaluate.

While the resulting fitness scores do not depend on the hardware the experiment runs on, the duration used on calculating them does. It ran on a water-cooled Intel i7-4790K CPU (8M Cache, up to 4.40 GHz)[1] with 8GB of RAM. The GE, SA and brute-force processes are parallelized and ran with 9 threads.

When sample groups are compared, their distributions are tested for normality using the Shapiro-Wilk normality test [41]. If Shapiro-Wilk is not applicable because of the sample size, a Q-Q plot is analyzed [42]. Robust Brown-Forsythe Levene-type test is used to test if the variances in the distributions are equal [43]. Finally, in the cases when normality cannot be assumed, Mann-Whitney U test is used to test the location shift of the distributions [44].

Grammatical Evolution Parameters

GE, as with GA, depends on multiple parameters, and with tournament sampling there are even more. Therefore, before evaluation the GE process, good parameters should be found. Additionally, seeing what parameters are needed for a good performing GE may reveal additional properties of it. The parameters were found using a combined manual and automated approach by searching the parameter

[1] Intel i7-4790K: `https://ark.intel.com/products/80807/Intel-Core-i7-4790K-Processor-8M-Cache-up-to-4_40-GHz`

space in multiple steps. The idea was not to find the optimal parameters, but rather to find some parameters that made the GE produce an individual with good fitness within a reasonable time. It was assumed that the size parameters—population size and number of generations—are the baseline parameters for GE and thus should be found first. Then the tournament size for the tournament selection strategy should be found. Then good parameters for the recombination rates—crossover rate and mutation rate—should be found. Finally the parameter for the GE gene duplication operator should be found.

To search for the GE parameters, a breadth-first graph search inspired approach was used. An example is shown in Figure 6. The approach tests a combination of parameters by doing 20 GE runs with those parameters and using the mean fitness score of the best individual from each run. If the score is an improvement of the previous score, the node is considered an improvement and it will consider the neighboring nodes in the graph by increasing the parameter values. Otherwise, the current parameter values are considered as not improving the performance and thus be discarded. When there are no more improvements, the parameter values that produced the best score is returned. The parameter values start at a small value and increase in an exponential fashion. This is based on the assumption that the parameters are more sensitive when the values are small. All of the parameter searches, except for the reproduction parameter and duplication rate search, are increasing with a multiplicative of 2. Because the reproduction and duplication rate parameters are bounded ($[0, 1]$) and to cover the whole range with a reasonable amount of steps, custom steps of 0.01, 0.1, 0.5 and 1 are used.

Grammatical Evolution Test

The reason for using GE is to search the space in an efficient way. To see if it is doing this, it needs to be compared to a brute-force approach that finds completely random samples in the space and picks the best of them. Thus, the hypothesis is that DGEL GE finds plants with better fitness scores than brute-force does (**H1**).

To make a fair comparison, the brute-force should generate as many individuals as the GE generates and modifies (through mutation or crossover). This way, both methods "generate" the same amount of individuals, either being completely new individuals or modifications of existing individuals. GE uses a fixed number of generations and population size, resulting into a fixed number of total individuals: $individuals = population + population * generations$, that the brute-force method also generates.

Because both methods depend on a random seed, an average of multiple runs is used. A sample size of 11 was selected as balanced choice between sample size and duration.

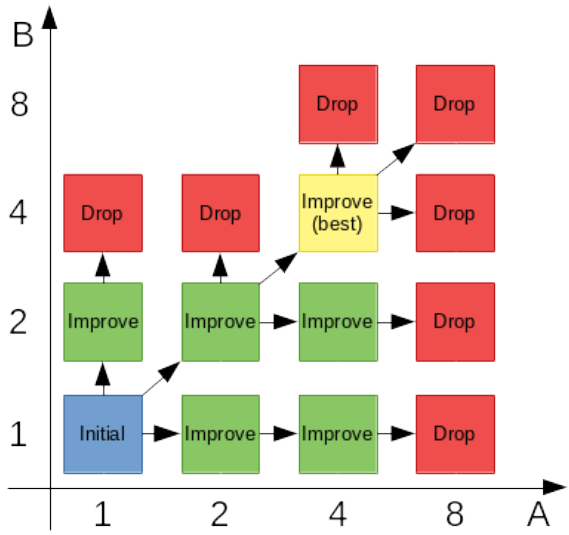


Figure 6: Example of a GE parameter search

Simulated Annealing

The main purpose of using SA with a grammar distribution is to improve the efficiency of the remaining components of DGEL by focusing the search space on a good area. Based on this a hypothesis can be formulated: An SA-optimized grammar distribution can find better individuals than a uniform grammar distribution (**H2**). An important word in the hypothesis is *can*, because this means that the hypothesis does not imply that *all* SA-optimized grammar distributions find better individuals.

SA is used as described in previous sections to find an optimized grammar distribution, and the progress of it is plotted to see if it makes any meaningful progress. Then, to test the hypothesis a comparison is made between the average fitness of L-systems randomly generated by a uniform grammar distribution and L-systems randomly generated by an SA-optimized grammar distribution. The L-systems are generated in a brute-force manner, not using any evolutionary approach such as GE.

If **H2** is supported, we hypothesize that random brute-force with an SA-optimized distribution can perform as well as GE with a uniform distribution (**H3**). This is tested by the same method as for **H1**, but with random brute-force using an SA-optimized grammar distribution instead of a uniform grammar distribution.

Additionally, we can hypothesize that GE with an SA-optimized grammar distribution can find better individuals than with a uniform grammar distribution (**H4**).

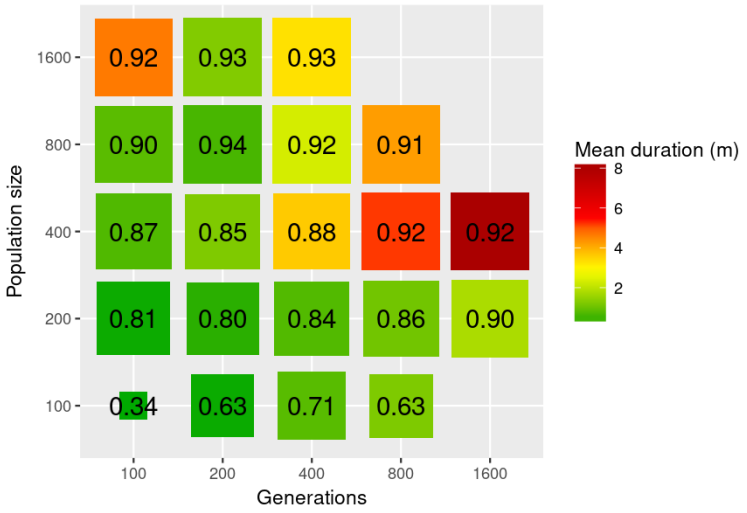


Figure 7: Visualized search of GE population size and number of generations. Numbers are the mean of best fitness scores, box sizes are a representation of that number, and colors represent the mean duration of the GE.

As shown in Section 5.1.2, GE is already performing so well that adding SA can not improve it further. Therefore to make a fair comparison, the number of GE generations is halved (from 200 to 100). Other than that the comparison is performed in the same way as with **H1**, but with brute-force replaced by GE with an SA-optimized grammar distribution.

5.1.2 Results

Grammatical Evolution Parameters

Figure 7 shows the search of the population size and number of generations to be used for the GE process. The search was prematurely stopped because when either the number of generations or population size reached 1600 the duration became unreasonable long. As seen, a minimum of either a population size of 200 or 200 generations is required for a reasonable performance. Additionally, for any number of generations, a population size of 200 makes a notable improvement. From this point on, there is a steady improvement from scores around 0.8 to 0.9 with both parameters increasing towards 1600. Outside a line drawn through 800, 400 and 400, 800 there does not seem to be any notable improvement, while the duration increases by a large amount. The parameter values that yielded the best fitness was $generations = 200$ and $populationsize = 800$, and they did it within a reasonable duration. The duration of it is significantly smaller than other parameter values that yield approximately the same fitness. Thus these parameter values were

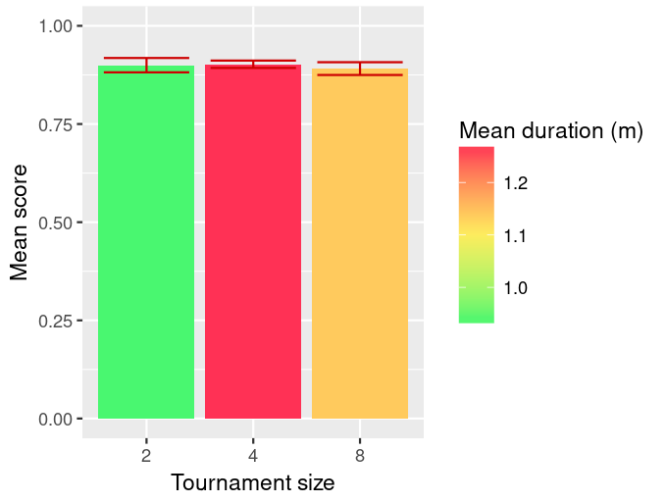


Figure 8: Visualized search of the GE tournament selection tournament size with standard error

selected for use.

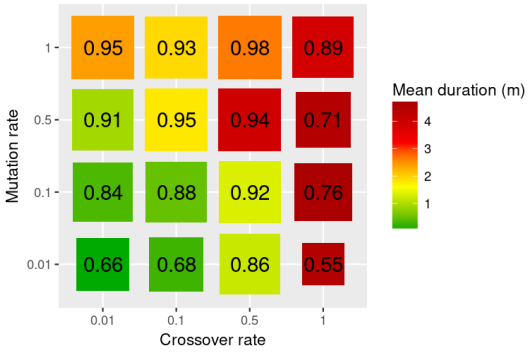
Figure 8 shows the search of the tournament size. Contrasted to the other searches, this is a one-dimensional search. As seen in the figure, the search did not find any improvements. Therefore the tournament size was kept at the lowest value, i.e. 2.

Figure 9a shows the search of the recombination parameters: mutation rate and crossover rate. There is a general improvement in fitness with both increasing mutation rate and crossover rate until the crossover rate reaches 1 where it suddenly turns for the worse. Therefore the crossover rate should at least be below 1. Additionally, the crossover rate seems to be the parameter that contributes the least to an improved fitness and increases the duration the most. Therefore the mutation rate should be the most important parameter. The best fitness score is provided by a mutation rate of 1 and crossover rate of 0.5, further indicating this.

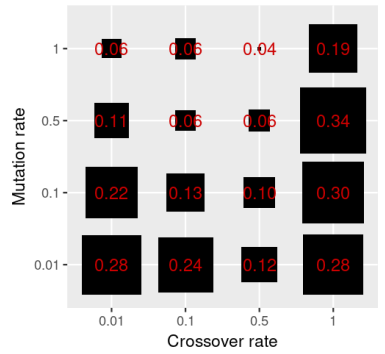
Figure 9b shows the standard deviations in the same search. The pattern is similar to that of the mean fitness scores. The most stable performance is provided by a mutation rate of 1 and crossover rate of 0.5. Thus the recombination parameters is set to these values.

Figure 10 shows the search of the duplication rate. It shows a worsening of both score and duration from a rate of 0 to 0.1, from where it stays bad. Therefore the duplication rate is set to 0.

All of the GE parameters are summarized in Table 9.



(a) Mean of best fitness scores as numbers and sizes, with duration as colors



(b) Standard deviation

Figure 9: Visualized search of the GE recombination parameters

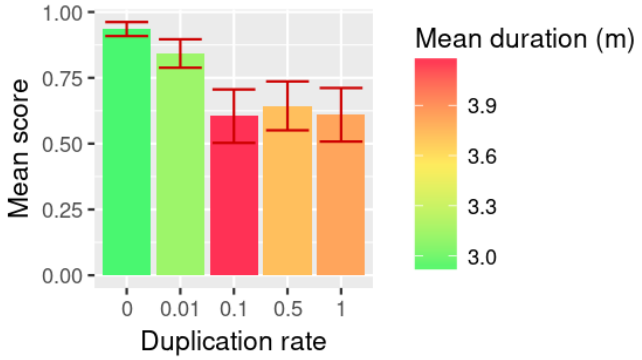
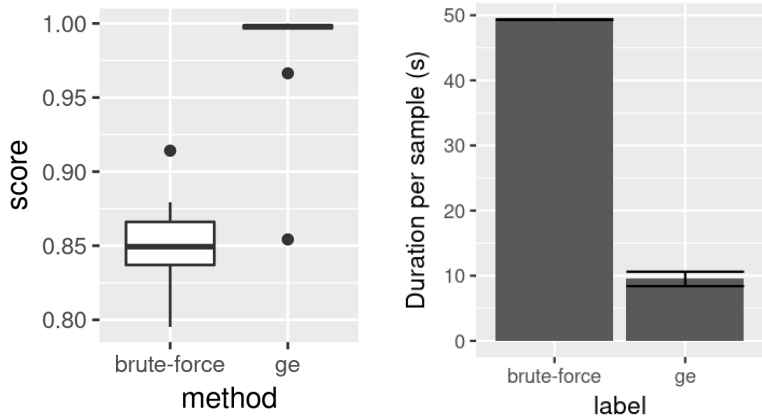


Figure 10: Visualized search of the GE duplication rate with standard error

Parameter	**Value**
Population size	800
Generations	200
Tournament size	2
Mutation rate	1.0
Crossover rate	0.5
Duplication rate	0

Table 9: GE parameter values selected based on searches



(a) Box plot of scores

(b) Mean duration with SE

Figure 11: GE and random performance compared

Grammatical Evolution Test

With a population size of 800 and 200 generations, the total number of individuals generated will be 160,800 (according to Section 5.1.1). Figure 11a shows the comparison of GE against random samples. A Shapiro-Wilk normality test suggests that the brute-force sample distribution can be assumed to be normally distributed ($p >= 0.05$, $W = 0.97$), while the GE sample distribution can not ($p < 0.05$, $W = 0.46$). A robust Brown-Forsythe Levene-type test suggests that the variances of the distributions can be considered equal ($p >= 0.05$, statistic = 0.09). Because there are non-normal distributions and different variances, the Mann-Whitney U test is used. GE has a median score of 1.00, while random has 0.85 (difference of 0.15). The Mann–Whitney U test suggests that the true location shift of the distributions is greater than 0 ($p < 0.05$, $U = 117$).

Additionally, Figure 11b shows the mean duration of the methods where GE is about 5 times faster than random. A Shapiro-Wilk normality test suggests that both the brute-force and GE samples can be assumed to be normally distributed ($p >= 0.05$, $W = 0.89$ and $p >= 0.05$, $W = 0.93$ respectively). A robust Brown-Forsythe Levene-type test suggests that the variances of the distributions can be considered unequal ($p < 0.05$, statistic = 17.13). Because the distributions can be assumed to be normally distributed and the variances can be assumed to be unequal, the Welch's t-test is used. GE has a mean duration of 9.51, while random has 49.33 (difference of -39.82). The Welch's two-sample t-test suggests that the true difference in means is not 0 ($p < 0.05$, $t = -35.90$, $df = 10.02$)

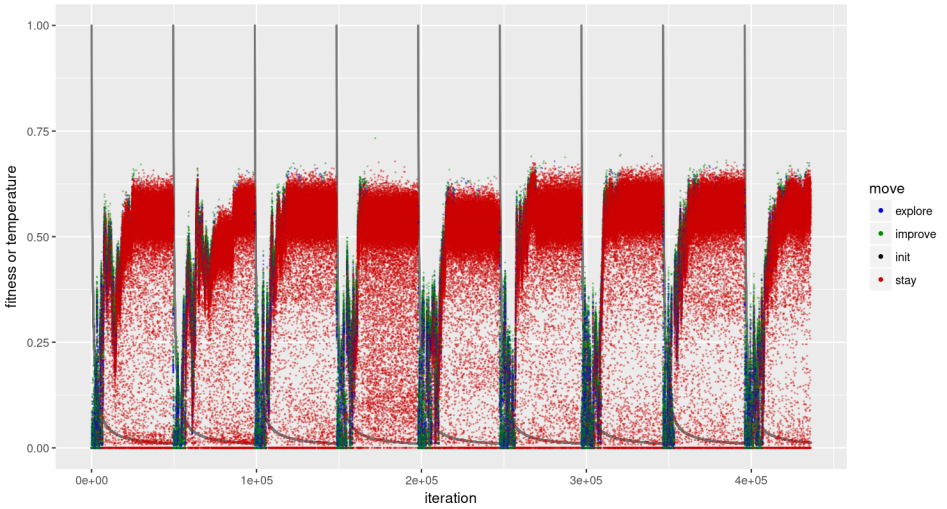


Figure 12: The complete SA process with multiple re-annealings

Simulated Annealing

Figure 12 shows the progress of one SA run, and Figure 13 shows a close-up of the first annealing. The run had a maximum of 2000 moves per dimension and a cooldown rate of 0.002. The grammar distribution evaluation had a minimum 32 samples and an error threshold of 0.004. In total this resulted into 436000 moves over 9 annealings. As seen in the figure, for each annealing there is a general improvement from a fitness score of 0 to around 0.6. The SA process explores the space in the beginning, moving both up and down in score, and then gradually focuses more on hill climbing towards the end. Each annealing generally reach a plateau before getting halfway through the annealing process, from which point nearly all mutations are worse and so it chooses to stay, though the second annealing is slower to reach the plateau. The fourth annealing found one exceptionally good grammar distribution with a score of almost 0.75, but when the same grammar distribution was measured over 100,000 samples its score was 0.61 (SD = 0.25). Therefore the score measured in the SA process may have been inaccurate. As this distribution was the one with the best score during the SA process, it is the one used for further analysis.

Seen in Figure 13 the points cluster around the same score. At the same time there are some points spread fairly uniformly between the clustering and 0. Additionally there is a clustering at exactly 0.

As the sample size (n = 100000) is too large for a Shapiro-Wilk normality test, a Q-Q plot is used. It can be seen in Figure 14, and it is clear that the distributions

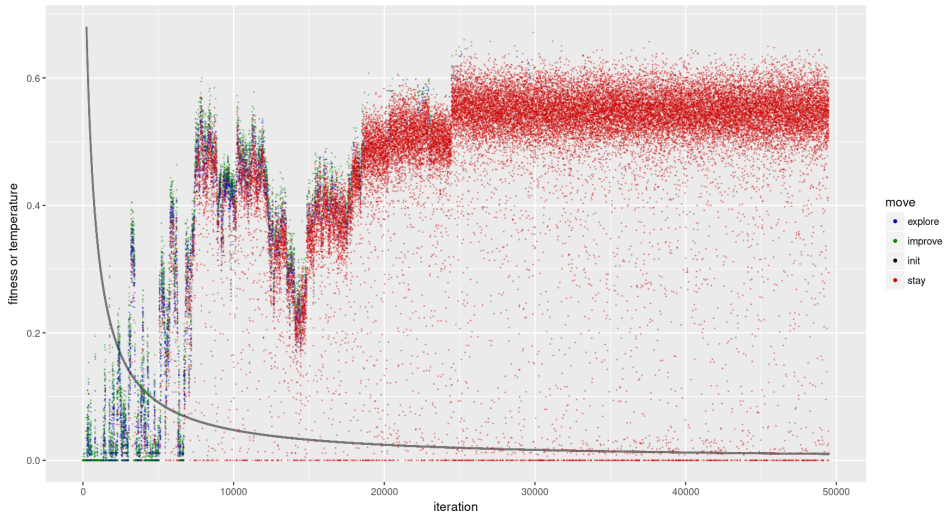


Figure 13: Close-up of the first annealing of the SA process

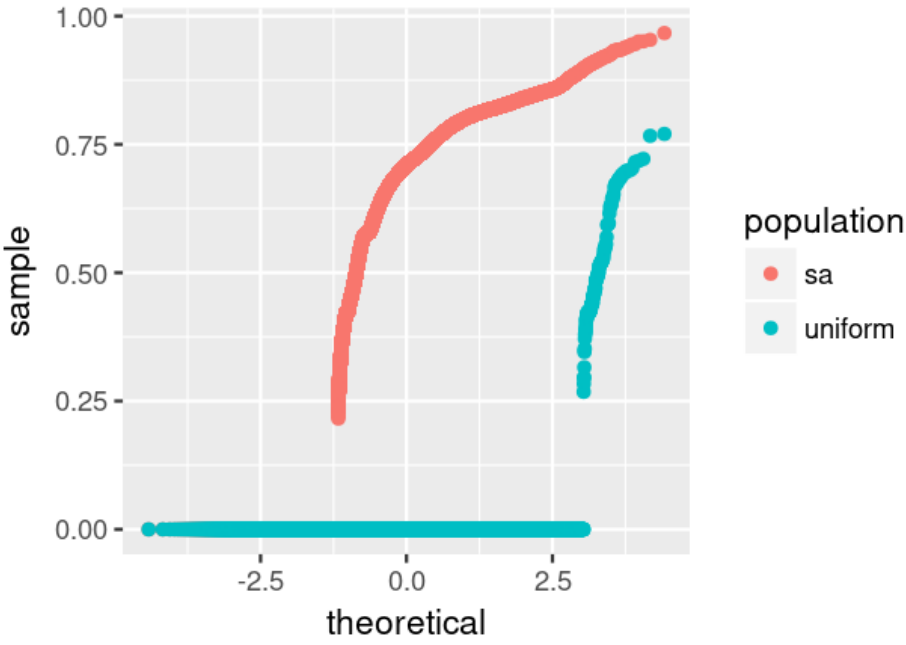
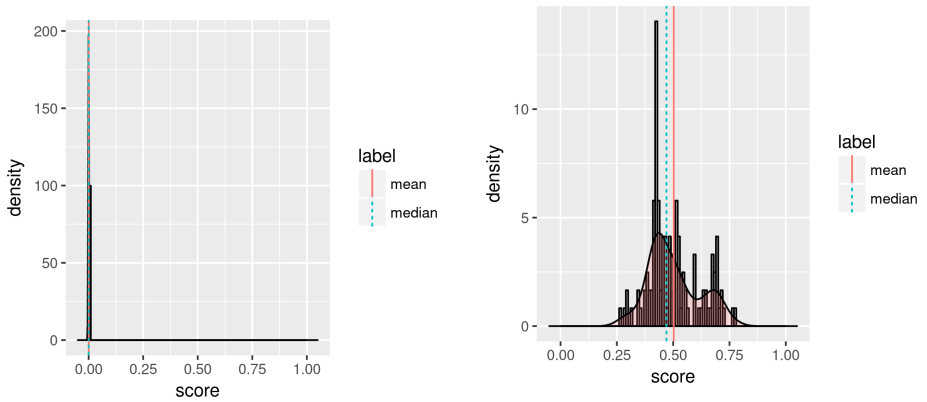


Figure 14: Q-Q plot of SA and uniform sample populations



(a) Complete sample population

(b) Sample population without samples with score 0

Figure 15: Sample population of L-systems generated using a uniform grammar distribution

are not normally distributed. Even with all 0 scores removed, the data does not appear normally distributed. A robust Brown-Forsythe Levene-type test shows that the variances of the distributions can not be considered equal ($p < 0.05$).

Figure 15a shows the uniform grammar distribution' sample distribution, where practically the whole population has a score of 0. When removing all zero-scored individuals, another distribution presents itself in Figure 15b. Here, a somewhat double bell-shaped distribution around the score 0.5 with a high concentration at around 0.4 is revealed for the remaining 121 individuals (0.12% of the total). Figure 16 shows the SA grammar distribution's distribution of fineness scores. 87.87% of the SA sample population have a score larger than 0, compared to 0.121% in the uniform sample population.

Figure 17 shows a comparison of L-system populations generated by the grammar distribution found by SA in Figure 12 and a uniform grammar distribution. The SA grammar distribution has a median of 0.71, compared to 0.00 for the uniform grammar distribution. A Mann–Whitney U test on the full sample suggests that the true location shift is not zero ($p < 0.05$, $U = 608550000$). Even with all individuals with score 0 removed, the SA grammar distribution's median score is higher with a median of 0.72 compared to 0.47, though the difference of 0.25 is smaller. A Mann–Whitney U test on the sample without zero scores still suggests that the true location shift is not zero ($p < 0.05$, $U = 1318100$).

Figure 18 shows the comparison of random brute-force with an SA-optimized distribution (bf-sa) against GE with an uniform distribution (ge-uni) (**H3**). A Shapiro-Wilk normality test suggests that both bf-sa and ge-uni samples can not be assumed

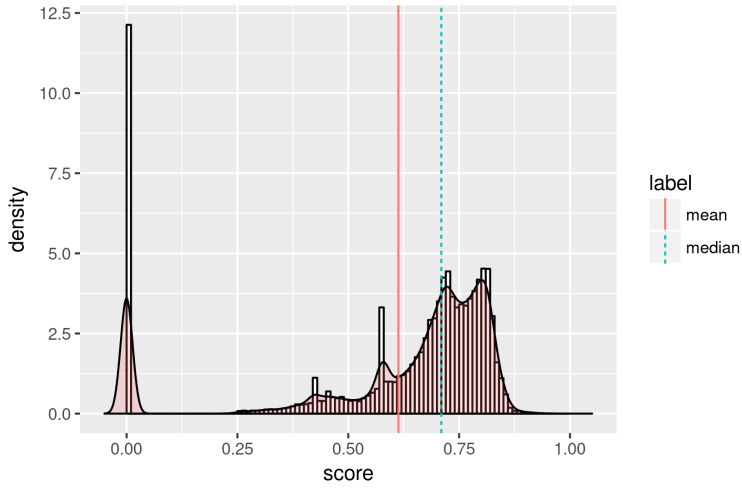


Figure 16: Sample population of L-systems generated using the SA-optimized grammar distribution

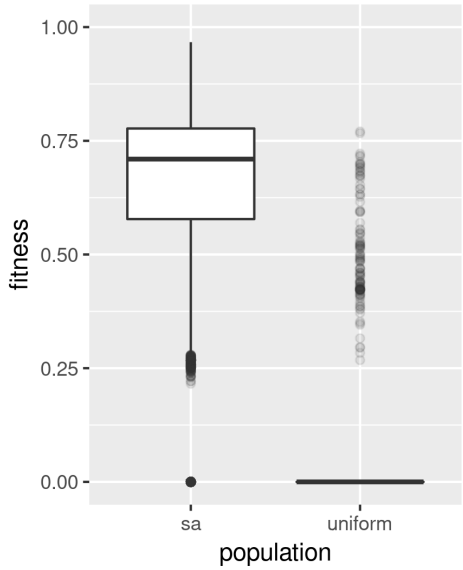


Figure 17: Box plot of random population with uniform grammar distribution compared to SA-optimized grammar distribution

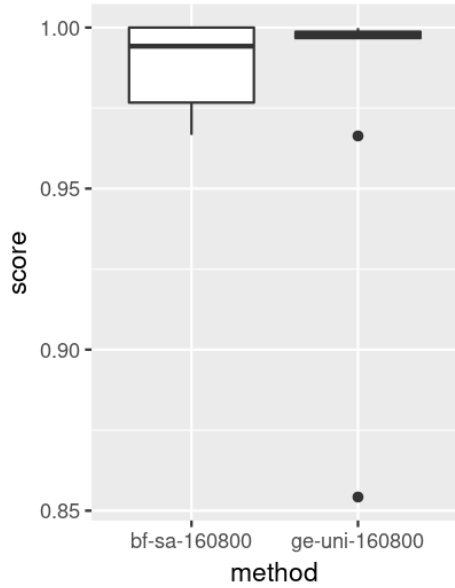


Figure 18: Box plot of random brute-force with SA-distribution compared to GE with uniform distribution

to be normally distributed ($p < 0.05$, $W = 0.81$ and $p < 0.05$, $W = 0.46$ respectively). A robust Brown-Forsythe Levene-type test suggests that the variances of the distributions can be considered equal ($p >= 0.05$, statistic = 0.17). bf-sa has a median score of 0.99, while ge-uni has 1.00 (with difference of 0.00). The Mann–Whitney U test suggests that the true location shift of the distributions is not greater than 0 ($p >= 0.05$, $U = 54$).

Figure 19 shows the comparison of GE with an SA-optimized distribution (ge-sa) against GE with a uniform distribution (ge-uni), both with half the number of generations (i.e. 100) (**H4**). A Shapiro-Wilk normality test suggests that both ge-sa and ge-uni samples can not be assumed to be normally distributed ($p < 0.05$, $W = 0.86$ and $p < 0.05$, $W = 0.60$ respectively). A robust Brown-Forsythe Levene-type test suggests that the variances of the distributions can be considered equal ($p >= 0.05$, statistic = 2.90). ge-sa has a median score of 0.99, while ge-uni has 0.86 (with difference of 0.13). The Mann–Whitney U test suggests that the true location shift of the distributions is greater than 0 ($p < 0.05$, $U = 118$). Additionally the box plot shows that ge-uni had some runs that scored 0 (to be exact, two runs).

The grammar distribution found in the SA process is shown in Figure 20. As seen in Figure 20a, this grammar distribution favors around 10 production rules in the L-system, though 20 productions is also a strong contender. Figure 20b shows

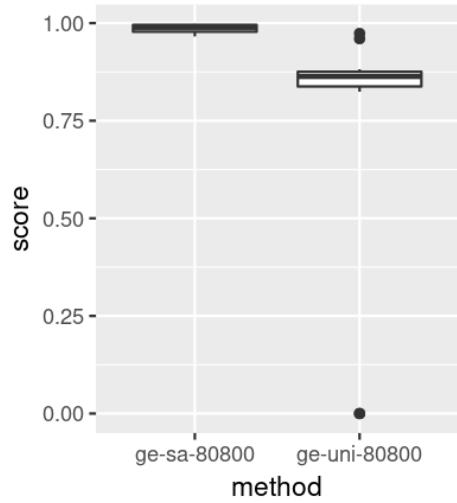


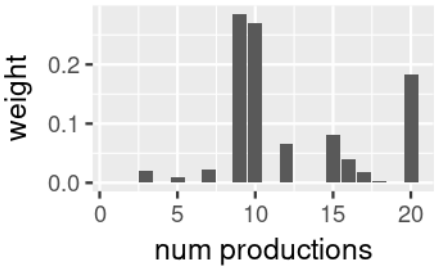
Figure 19: Box plot of GE with SA-distribution compared to GE with uniform distribution, with 100 generations instead of 200

that it always favors symbols over stacks, starting at around 60% symbols in the top depth, then around 90% symbols in the next depth, and 100% symbols in the final depth. This makes all of the distributions at depth 3 irrelevant as they will never be used. It favors shorter strings in depth 0, but not too short strings, as seen in Figure 20c. While in depth 1 and 2 it favors medium short and medium long strings.

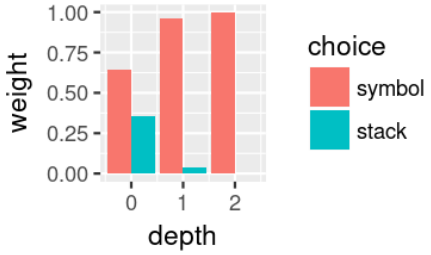
Figure 20d shows that it only allows variables in depth 0, while its nearly the inverse in depth 1 and 2. This is interesting as an operation has no effect if it is not followed by a variable. Thus depth 2, along with depth 3, is not contributing anything to the plant, but depth 2 is still wasting instructions. Depth 1 also has only about 10% chance of a variable, meaning that most of the drawing will happen in depth 0, which in turn means less branching.

The operation distribution in Figure 20e has a strong tendency of yawing to the right (-) than the left (+) in depth 0. In depth 1 it barely allows yawing at all, while in depth 2 it is the opposite with a strong focus on +. The other rotations (pitch and roll) also have one-sided tendencies, but not as strong. Among all of the 20 variables available, `F` is the only variable that is an instruction, the draw line instruction. Figure 20f shows that it is included and is in fact one of the stronger variables in the distribution. Only 3–4 of the variables have a strong weight in each of the depths.

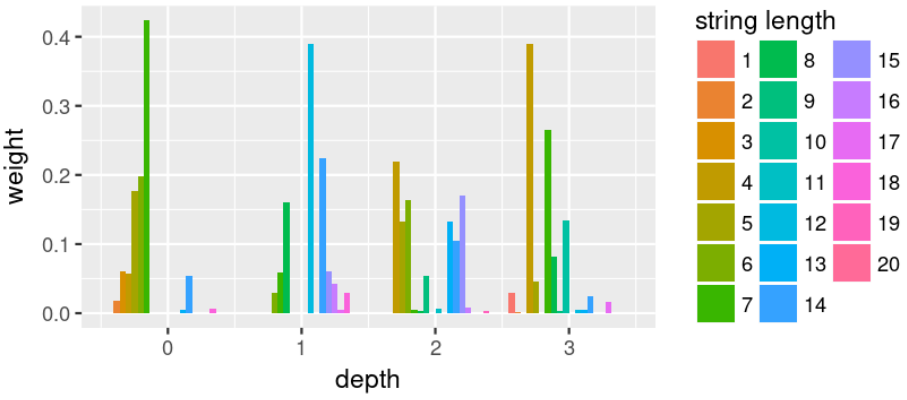
An example of a plant generated with the SA-optimized distribution can be seen



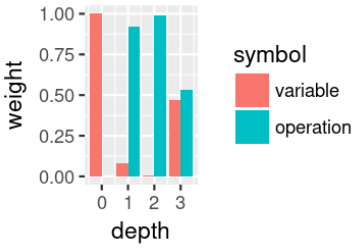
(a) 1*20production



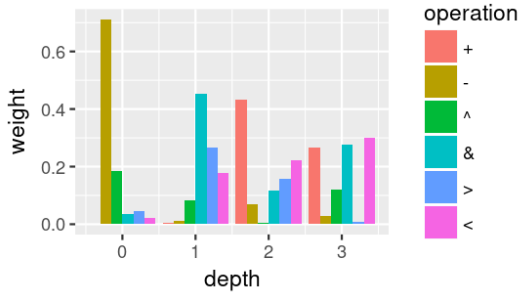
(b) symbol / stack



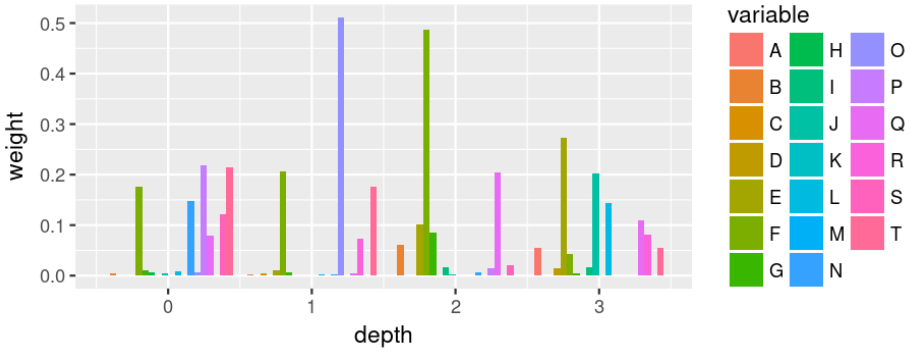
(c) 1*20(symbol / stack)



(d) variable / operation



(e) "+" / "-" / "^" / "&" / ">" / "<"



(f) %x41-55

Figure 20: SA-optimized grammar distribution. Captions are the respective parts of Listing 1.

Figure 21: Example plant generated by the SA-optimized grammar distribution, fitness 0.78

```
axiom: NT[>>&&&&>&&&F[++++<<]]P[<<&^[>-+<+>]&><&&&&&O]P
F -> Q[O&^&&&^&<>O[<<+++++^>++<++&][++++-]&>]NT[^>R
    >&>[<<&<+&+&<+&+-<]<<>&&]P
L -> [O&<<>&^&&&&>]S[^<[+++-<<>>++<<+<-]^>&<<&>&&>F^>]T
    [<<&><^&&O&<>]
N -> N[><&<^&<<&&^&&T
    ][&>[+-<<>+++&+&<+][+++++&><++><<^-]&&&>]PN[&&>&>T
    &&&&^&><<&]
P -> [&&&O>&>&&>>&&F]F[<&&&&&&>&&&&]TF
S -> [><&<&&>&]TSQS[&T<<<>[>++-+]<<&&>]T
T -> PT[>&^>&&&>&&R&&<>>]
```

Listing 3: L-system representation of plant in Figure 21

in Figure 21, and its L-system is found in in Listing 3. This plant is very tall and straight, has few branches and few fractal patterns. The L-system goes to depth 2, but depth 2 never includes any symbols, and is therefore not relevant for the 3D model. Depth 1 is mostly producing operators, but also an occasional variable. There is recursion in the segment drawing as `F` produces `P` and `T` which in turn produce `F`. Additionally, part of the recursion is inside a stack, thus allowing for fractal patterns.

5.2 Evaluation of Evaluation Module

5.2.1 Method

RQ2 asks "how aesthetically pleasing plants can be generated". DGEL was the developed solution to this question, but to know if it can actually generate aesthetically pleasing plants or not, the evaluation module has to be analyzed and compared to how humans rank the plants. If the ranking made by the humans agree with the ranking made by the fitness component in the evaluation module, it would suggest that the generator knows how to distinguish good plants from bad plants, and can apply its evolutionary algorithms to generate aesthetically pleasing plants. So the hypothesis (**H5**) is: Humans agree with the ranking made by the DGEL fitness component. If the humans do not completely agree with the DGEL ranking, the next question is why this is the case and what can be done to improve the rank correlation?

While the first question can be answered by quantitative methods, the second question requires support from a qualitative analysis to understand what factors are important to distinguish good generated plants from bad. Thus an embedded design mixed-method with qualitative data as a supplementary is used [45].

A pairwise comparison of a set of DGEL-generated plants is performed by hu-

man participants and the AHP is used to create a ranking of all of the plants [46]. By using analytical hierarchy process (AHP), we do not only get a ranking of the plants, but also distances between the ranks. This is useful because humans may consider some of the plants equally pleasing, or some plants much more pleasing than others. Finally, all of the rankings are aggregated and compared to the ranking made by DGEL using Kendall's Tau [47], and dRank [48]. Because the human rankings only has priority weights and not absolute scores, the DGEL fitness scores are translated into weights by dividing each score by the sum of the scores such that they sum to 1. The data fit the requirements of Kendall's Tau [49]: they are unweighted, conjoint and have a definite range. Therefore Kendall's Tau is used to test the ranking correlation. It also gives an indication of how correlated they are in range $[-1, 1]$, where 1 is perfectly positively correlated, 0 is not correlated and -1 is perfectly negatively correlated. But there are two problems with this: it does not consider the relative distances between the ranks [49], and the different participant's rankings have to be aggregated beforehand, meaning Kendall's Tau does not consider the variance between the different human rankings. Therefore, dRank is used as an additional statistic, as it considers the distances between ranks and that there are multiple rankings aggregated together [49, 48]. Whenever a dRank statistic is shown, the B is the bootstrap sampling size used for estimating the p-value [48].

As many plants as possible is desired, but humans have limited patience and the number of comparisons to rate is $\frac{n*(n-1)}{2}$, where n is the number of plants. Therefore a limited set of plants has to be used. The final number of plants used is based on the estimated time used on a comparison and the estimated time a participant is patient. The plants should also be spread over the whole range of the fitness score, so that the rank comparison will be valid for the complete range. They should also always be *something* (have at least one branch), otherwise there is nothing to rate. The ideal score range would be $[0, 1]$, but it was found difficult to generate plants with perfect 0 (with the plant being *something*) and 1 scores. Thus GE is used to generate the worst possible plant with score above 0, and the best possible plant. Then, $n-2$ additional plants are generated uniformly between these two extremes.

To estimate the number of plants to rank, and to assess the quality of the survey, a three-step process is used. First, a rough estimate is made based on researchers intuition and an assumed patience of 10 minutes. Then, in the second step the researcher performs the pairwise comparison while measuring the time used. Based on this and the bias of the researcher, a new estimate is created. The third and final step is to run two or more pilot runs on another person with the previous estimates. The time used is measured, their actions are observed and they are asked some

questions at the end. They were asked to complete the survey as described in its introduction, think aloud, tell us when they get tired and answer some questions at the end. Based on this, the number of plants is re-estimated with a margin in case of bias, a new set of plants is generated, quality issues are fixed, and the pilot is run once more to confirm the new estimate.

Participants were sampled using convenience sampling and snowball sampling. As such, the results may not be generalizable to the whole population, but they may be used as an indication as to what is important in an aesthetically pleasing plant. Convenience sampling was selected because of time and resource restrictions, and snowball sampling was added as a means to gather further samples. Participants were found by asking friends, family and acquaintances either by direct conversation or by posting publicly on social networks including Facebook, Google+ and Twitter. The participants were asked to share the survey to others, and a sharing link was shown at the end of the survey, thus allowing for a snowballing effect. Before the survey link was shared with others, four people were observed directly while taking the survey, with a method similar to that of the pilot runs. This both allowed for deeper qualitative data and final adjustments to the survey.

5.2.2 Survey Design

The survey consists of a consent form, a pre-questionnaire, a description, the pairwise comparison, a post-questionnaire with the results, and finally a thank-you page with the sharing link. The pre-questionnaire is intended to map the demographic and what relation they have to plants and games. The description describes the pairwise comparison task, what they should do in it, and what to expect at the end. The post-questionnaire is intended to find out how much the participant agrees with the ranking calculated by AHP or why they do not agree, and what they find important in plants. Finally, the thank-you page is used to assure the participant that they have completed the survey, give them the option to see or share their results, and share the survey to others.

In the pre-questionnaire the participants are first asked about their age, gender, education and occupation. They are then asked “how often do you work with plants? Including all types of interactions with plants, such as gardening, household plants, photography, etc.”, and “how much do you like plants in general?” on a 5-point Likert scale. Their purpose is to see if their relation to plants affects how they rank the plants. Finally, they are asked “how often do you play video games?” on a 5-point Likert scale. This is to see if their relation to video games affects how they rank the plants. A person that regularly works with plants or enjoys looking at plants in nature may expect something different in a plant than a person who regularly sees plants in video games.

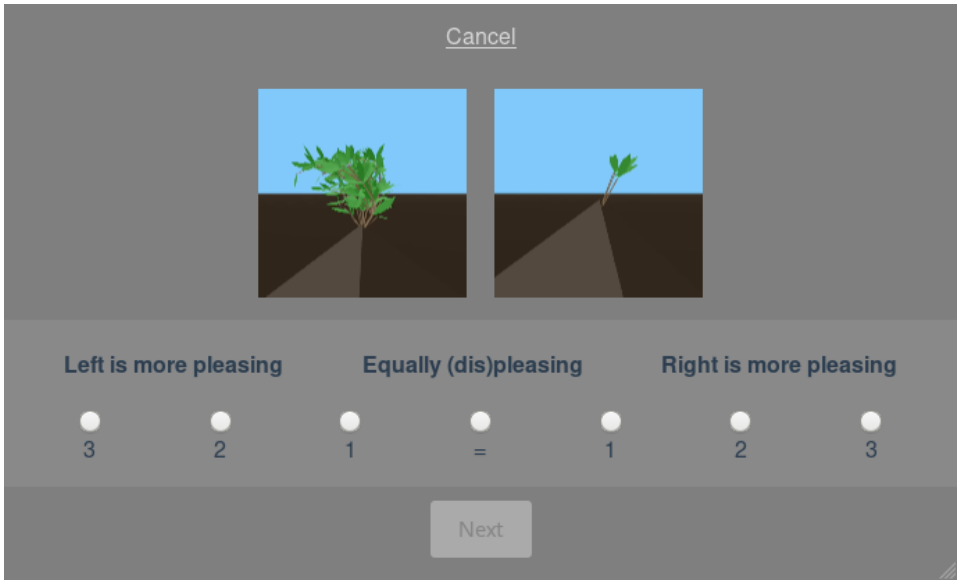


Figure 22: Pairwise comparison task as viewed on a small screen

During the description of the task, they are recommended to enter fullscreen so that they do not have distractions and so that the plants can be viewed as big as possible. They are also asked to try a different browser or withdraw if the plant visualization is not working as expected. This is because the quality of the visualization may affect their ratings.

Figure 22 shows the comparison task. It is kept as simple as possible to avoid distractions, and a neutral gray background color is used to avoid the perceived hue or brightness of the plants to be shifted. Two plants are visualized side by side as looping videos of a camera rotating around the plant so that all sides of the plant is seen. In addition, the camera adjusts its height and distance from the plant so that all of the plant can be viewed. The rotation happens at a balanced speed that lets the user view all of the plant in a short amount of time, while being slow enough to let them study it.

The video runs at 60 frames per second (FPS) to give a smooth experience, to match the monitor refresh rate and to match what is commonly used in games. They are encoded as both VP9 WebM and H.264 MPEG-4 to support a wide range of browsers. According to Mozilla Developer Network (MDN), H.264 is supported in most browsers, but depends on platform support, and their support may be removed as the format has patents make them “unfit for the open web platform” [50]. Therefore the videos are encoded in both formats and VP9 is the preferred one if both are supported.

They are compressed as much as possible to reduce load times, while still keeping a near lossless visual quality as to not bias the comparisons. Both are encoded with the FFmpeg tools[2]. Both the FFmpeg Wiki and Google Developers recommend using Constant Quality for WebM [51, 52]. They recommend CRF values from 15 to 35, and based on the resolution and Google Developers's resolution and CRF value matrix, a CRF of 32 is used [51, 52]. For H.264, according FFMpeg Wiki a CRF value of 17–18 can be considered "visually lossless" [53]. Thus, 18 was selected as it allows the files to be smaller than 17. Based on this, WebM is encoded with the parameters `-quality best -crf 32 -b:v 0`, while MPEG-4 is encoded with `-preset veryslow -crf 18`. `-quality best` and `-preset veryslow` enable better quality per file size at the cost of slower encoding [53], but as the videos are encoded offline, the encoding time cost is not important.

The participant is presented with three categories to choose between: left is more pleasing, they are equally (dis)pleasing, and right is more pleasing. If they find one more pleasing they need to rate how much pleasing it is on a scale of 1 to 3. The original AHP method uses an 8-point scale for each side [46], but this is too detailed for a "normal human", and so it was reduced to 3 points which allows for a low, medium and high rating. Based on feedback from the pilot run, the "equally (dis)pleasing" option was labeled "=" instead of "0" because "0" seemed like it meant that the plants were bad. "(dis)" was also added based on the feedback because without it it seemed like it meant that the plants were good. With these changes, the participant should be less reluctant to select "equal".

By pressing the "next" button, the participant is presented with new pairs until all have been rated. The pairs are presented in random order and with random placement (left or right), as to not make the order create a bias.

The post-questionnaire asks some questions and displays the resulting order of the plants. Figure 23 shows an example of the resulting order. The dots on the white line represents the relative distance between the plants, and the plants themselves are shown in order below. In this example, the plant ranked as the best has a larger distance to the second than the second has to the third. The participants are asked if they "agree with the ranking of the plants shown below", and if they do not "strongly agree", they are asked "why do you not strongly agree with the ranking?" This can help determine if the method of ranking the plants with AHP is valid. Next they are asked the most important question: "What would you say separates good plants from bad plants in the ranking?" This is useful to be able to do a qualitative analysis of why the human ranking does not match the fitness ranking, if they do not match. Finally, they are asked for "other comments" in case they have something useful to add that do not fit the other questions.

[2]FFmpeg website: `http://ffmpeg.org/`

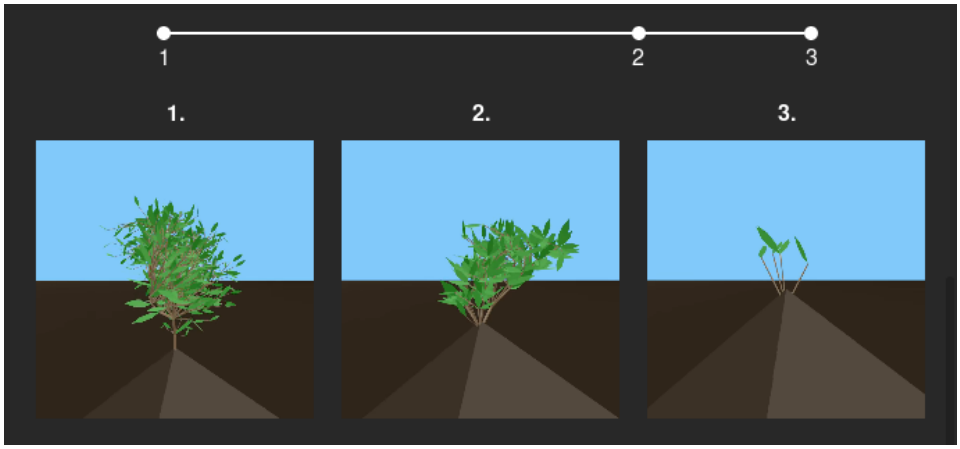


Figure 23: Example rank resulting from human rating of plant pairs

The final thank-you page contains a congratulation, a link to revisit their results, and a link to share the survey. This share link contains a token that indicates who it was shared from. With this data the snowball effect can be tracked. Though, there is nothing preventing them from sharing the original link, thus losing this information.

5.2.3 Implementation

No web-based application that matches the design of the survey was found. Thus a single-page web-application was created [54]. It uses the Vue.js JavaScript framework in the front end[3], the Rocket web framework for Rust in the back end[4], and MongoDB to store the data[5]. Multiple other libraries were used both in the front end and back end, but the aforementioned are the most central. As the application is a on-off application, only meant to be used for the survey, and fairly simple, the choices of technology and architecture were based on experience, convenience and preferences. Most of the technology used had recently been used in another project by the researcher, and thus allowed for code re-use and rapid development.

The application uses a client–server architecture where the server consists of two layers: API and DB. The client uses a Vue Component for each page described in the Survey Design, and URL routing is handled by Vue Router. The Components communicated directly to the API layer using a custom REST API, to for example register a participant or get the pairs of plants to evaluate.

The API layer is written in Rust and uses the Rocket framework to implement

[3]Vue: `https://vuejs.org/`
[4]Rocket: `https://rocket.rs/`
[5]MongoDB: `https://www.mongodb.com/`

the API. Multiple routes are defined based on the requirements of the client. It communicates directly to the MongoDB database using the *mongodb* Rust library.

When a participant registers, they receive a private token that is kept in the URL in the client throughout the whole survey. This token access to submit and retrieve their data. They also receive a public token that represents the participant, but does not grant access to their data. Thus, this public token is used in the share link to identify who shared it.

The participant participates only in a specific *task*. A task is a set of plants that are to be evaluated, and the system may contain multiple tasks. While only one task is used during the survey, allowing multiple tasks lets the researcher experiment with the application without affecting the real data while developing or during the survey.

The database contains three collections: *user*, *sample* and *weight*. *user* stores all data relating to the user and is uniquely identified by the private and public tokens individually. *sample* stores data about one of the plants, such as its name and the task it is part of. Finally, *weight* stores the rating submitted by one user for one pair of plants and is uniquely identified by the user and the two plants together.

Finally there is a command-line interface (CLI) available to extract the data either as raw data or as calculated priority weights using AHP. Because of this, a separate module *stats* in the API layer contains the common functionality used by both the API and the CLI.

5.2.4 Results

Plants

The lowest possible plant fitness score found by GE was 0.27, and the highest was 0.97. Thus, the plant fitness scores used are: 0.27, 0.34, 0.40, 0.46, 0.53, 0.59, 0.65, 0.72, 0.78, 0.84, 0.91 and 0.97. The plants will be referred to with their fitness score. All of the plants can be viewed in Figure 33 in Appendix A.

Participants

There were 56 people that registered. Of those, 37 completed ranking the plants. This means that 34% of the 56 participants did not rank the plant, of which many likely are people that unintentionally registered multiple times. Those who did not complete the ranking of the plants are excluded from the analysis. Finally, one of the 37 did not submit the post questionnaire.

The demographic of the participants is mainly 20–30 year old (65%) males (78%) with a bachelor or higher degree (76%) in information and communication technologies (49%). The age ranges from 21 to 69. Science & engineering and service & sales were also strongly represented with 16% and 14% respectively.

As seen in Figure 24, the participants tend to occasionally work with plants

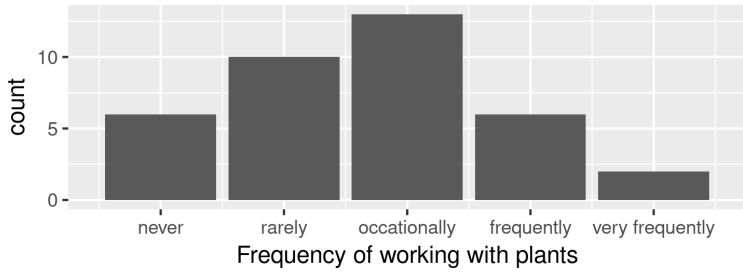


Figure 24: Participants' frequency of working with plants

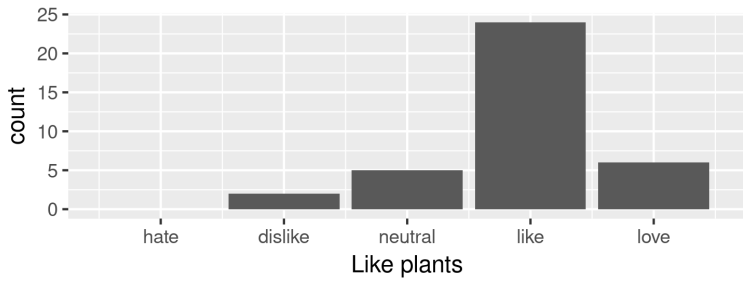


Figure 25: How much participants like plants

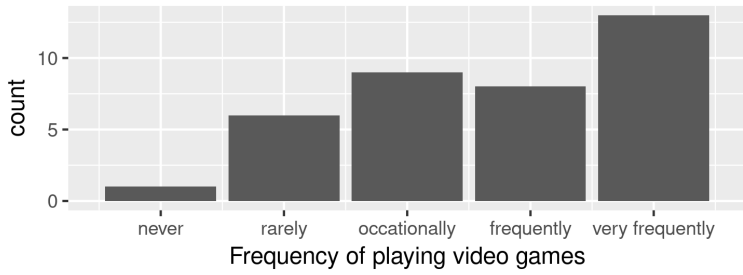


Figure 26: Participants' frequency of playing video games

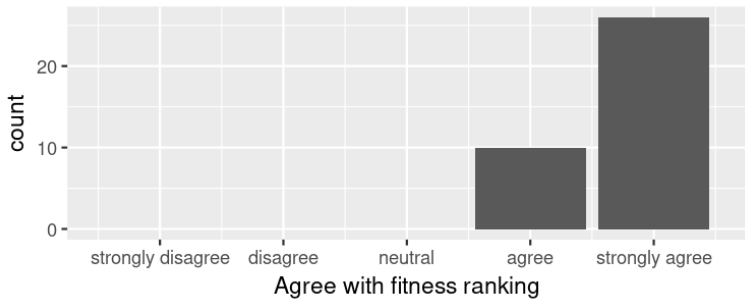


Figure 27: Human agreement with the AHP ranking

with a skew towards less frequently. Figure 25 shows that while the participants usually do not work much with plants, they tend to like them (64.87% like them and 16.22% love them), no one hates them, and only 5.41% dislike them. Thus the participants generally have a good relationship with plants. Finally, in Figure 26 there is a tendency towards playing video games frequently (median = frequently, 56.76% play at least frequently), and only 2.7% never play games.

There was only 1 participant that received the share link from another participant. Still, only around 25 people were directly asked to participate, and only 15 confirmed that they participated. Additionally, it is not expected that all of the people contacted directly did participate. Thus, at least 12 participants, and likely more, must have received the link from another person.

Ranking Agreement

As seen in Figure 27, the participants always agree with the ranking made by AHP. 73% strongly agree, while the rest agree. The ones that did not strongly agree have to explain why. These explanations were categorized into four categories based on the content: null, invalid, position and distance, as shown in Table 10. While the participants were obligated to write a reason, one participant was able to not write anything and is therefore categorized as *null*. Some explanations contained comments that were not answering the question, which could be because of misunderstanding the question, misinterpretation of the scale, or laziness. For example, the comment “I never agree strongly with online tests”, indicates that they use the scale differently, and actually completely agree with the ranking. Therefore, these comments were categorized as “invalid”. Finally, two categories were actual reasons to why they did not strongly agree: *position* and *distance*. A plant being positioned in a wrong rank was the most common reason, as it was commented by 5 of the participants. In all cases they only wanted to move one plant. Only one commented on the distance between the ranks.

Category	Description	Occurrences
null	No comment	1
invalid	The participant did not answer the question, but commented something irrelevant	3
position	Feels that a plant is in the wrong position and wants to change its rank	5
distance	Feels that the distances between some ranks are either too short or too long	1

Table 10: Reasons as to why 10 of the participants did not strongly agree with AHP ranking

Correlation Between Human and Fitness Ranking

The fitness ranking and the human aggregated ranking are shown together in Figure 28. From this it can be observed that both agree on the three worst and the two best rankings, while in-between these there are big differences. For example, 0.65, 0.72, 0.78 and 0.84 are ordered in the reverse order, and are far from the fitness ranking. 0.53 and 0.84 is also ranked as bad as the three worst by the humans, while the fitness ranks them higher. Finally, humans prefer 0.91 much more over 0.97, which is reverse of the fitness. Still they do somewhat agree with the ranking of 0.49 and 0.59, while they are reversed.

dRank calculates a ranking distance of 16.67 ($p = 0$, $B = 10000$), meaning that the human rankings are significantly different from the fitness ranking, and we reject the hypothesis that they are the same. Still, this does not mean that there is no positive correlation between the rankings, only that they are not equal. Kendall's tau is 0.55 ($p < 0.05$), indicating a positive correlation, though only at about 55%.

With the two best ranks (0.97 and 0.91) and three worst ranks (0.27, 0.34, 0.40) removed, thus looking only at the "middle" part, the dRank distance is 5.21, ($p = 0$, $B = 10000$), and Kendall's tau is -0.33 ($p >= 0.05$), indicating no correlation in either direction (*middle* in Table 11).

Analysis of Bad Correlation

Figure 29 shows the ordered human ranking. Both agree that 0.27 is the worst plant. 0.34, 0.40, 0.53 and 0.84 seems to be ranked as equally bad, just above 0.27. 0.78, 0.72, 0.59 and 0.49 seem to be slightly climbing in score. 0.91 is clearly in first place, while 0.97 and 0.65 are possibly on a shared second place. The standard errors are smaller on the bad plants than on the good plants. Both 0.84 and 0.46 were moved 5 ranks down and up respectively, almost swapping places. 0.84 is the worst, having a difference in weight between its neighbor of 0.12 compared to 0.05 for the 0.46 plant.

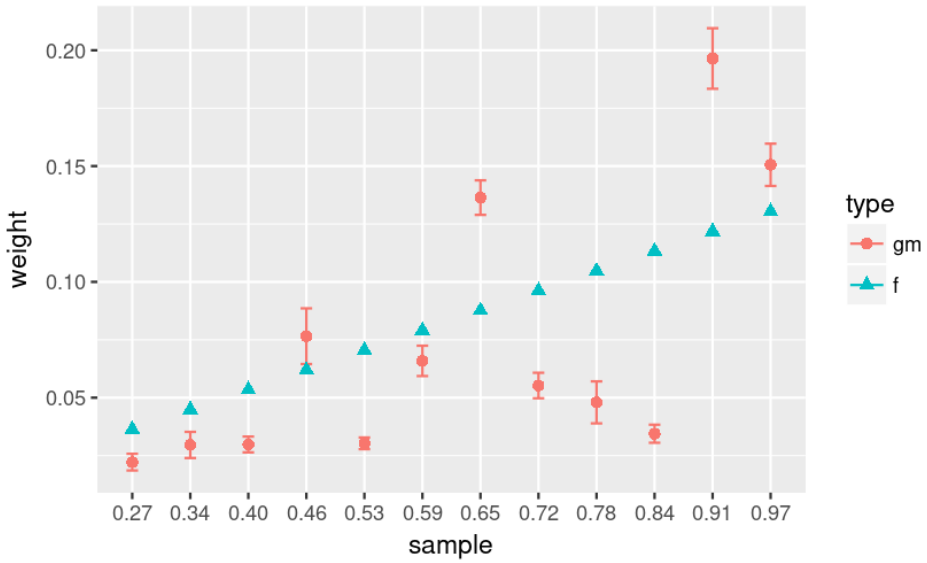


Figure 28: Normalized fitness score (f) plus geometric mean (gm) of human priority weights, with standard error

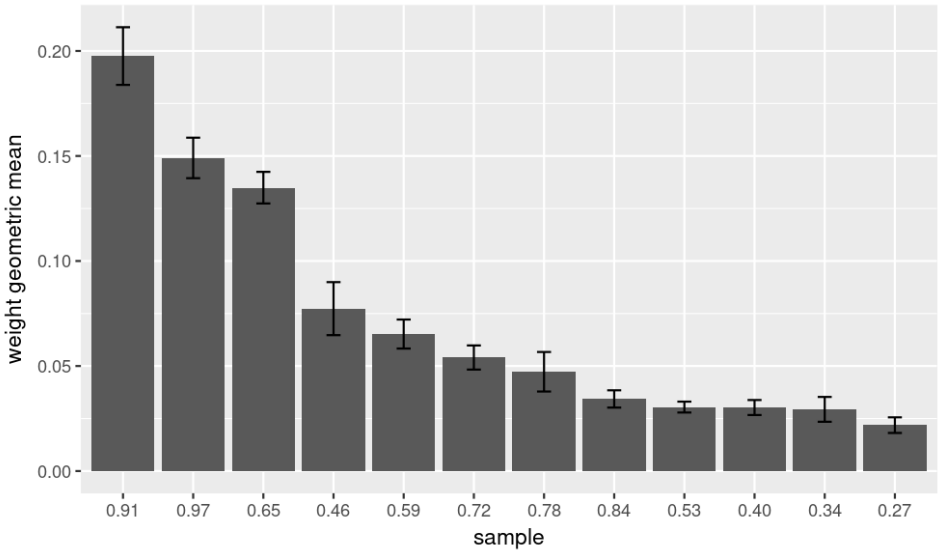


Figure 29: Priority weights from human plant ranking, with standard error

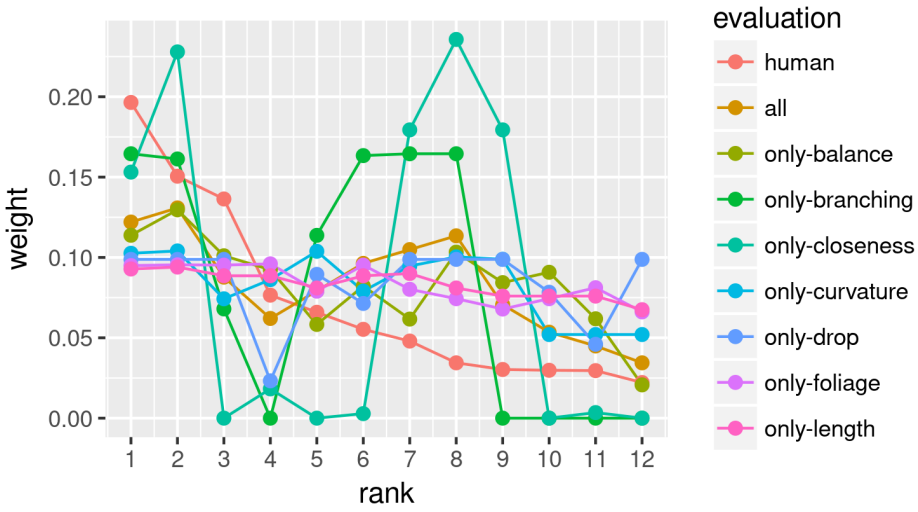


Figure 30: Human ranking compared to fitness ranking (*all*) and its components. The ranking on the x-axis is the human ranking.

Ranking	**Kendall's Tau**	**dRank**
all	0.55 (p = 0.01)	16.67 (p = 0, B = 10000)
middle	-0.33 (p = 0.88)	5.21 (p = 0, B = 10000)
only-length	0.75 (p = 0.00)	20.99 (p = 0, B = 100)
only-foliage	0.57 (p = 0.01)	17.01 (p = 0, B = 100)
only-balance	0.49 (p = 0.02)	16.67 (p = 0, B = 100)
only-curvature	0.45 (p = 0.02)	16.63 (p = 0, B = 100)
only-branching	0.36 (p = 0.07)	25.43 (p = 0, B = 100)
only-closeness	0.14 (p = 0.27)	16.63 (p = 0, B = 100)
only-drop	0.13 (p = 0.30)	21.67 (p = 0, B = 100)
removal	0.79 (p = 0.00)	16.67 (p = 0, B = 100)
removal-middle	0.52 (p = 0.07)	5.21 (p = 0, B = 100)

Table 11: Rank correlations between various versions of the fitness and the human ranking

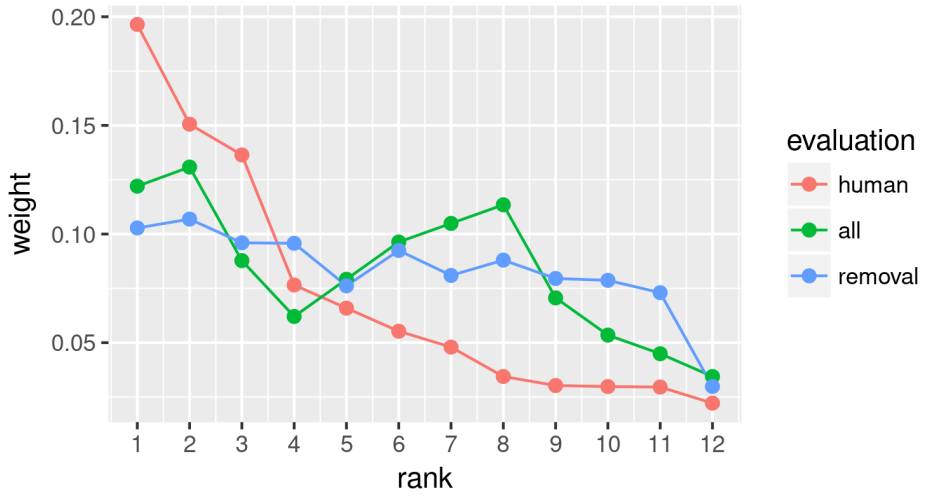


Figure 31: Human ranking compared to fitness ranking (all) and a version with branching, drop and closeness metrics removed (removal). The ranking on the x-axis is the human ranking.

Figure 30 compares the human ranking against fitness ranking and each individual fitness metric. The fitness ranking has a clear problem of scoring the ranks 5–9 too high, which creates a "bump" in the curve. Two metrics that stand out — closeness and branching — look like may be a major cause of this bump, as they themselves have an even larger bump. Additionally, the drop metric seems to be in a disagreement with the human ranking. These observations are also supported by Kendall's Tau and to some degree dRank, as seen in Table 11. Kendall's Tau indicates that all three metrics are not correlated ($p >= 0.05$). Branching also has a low correlation compared to the other metrics, but relatively higher than the other two. dRank supports that drop and branching have a low correlation, but disagrees with Kendall's Tau on the closeness ranking.

These findings together suggest that these three metrics may have a bad effect on the correlation of the human and fitness rankings. This is supported by the plot of the *removal* evaluation in Figure 31 and its rank correlation in Table 11 (*removal*). The "bump" has been removed, but there are instead some "waves" in the center part that disrupts the ranking. Additionally, there is less distance between the two best plants and the third best plant, and a larger distance between the worst and second worst plants. The Kendall's Tau is 0.79 ($p < 0.05$), which is an improvement of 0.24, while dRank indicates no difference. Additionally, the correlation in the middle region has also improved as seen by *removal-middle* in Table 11, though Kendall's tau is still not statistically significantly different from 0

($p >= 0.05$).

Human Perception of Aesthetically Pleasing Plants

All participants were asked openly what they would say "separates good plants from bad plants in the ranking". These comments were categorized into factors and the number of participants who mentioned them was counted. The factors were split into two: directional factors and directionless factors. Directional factors have a clear understanding of what is better and what is worse, while directionless factors only understand what is important without knowing which direction is better or worse. They can be found in Table 12 and Table 13 respectively.

The abstract category "natural" was mentioned the most (8 times). Many participants said that plants looking "natural" looked better, sometimes contrasted to "artificial" plants. The word "natural" was often used as a dependent factor of other independent factors. For example a comment said "[...]the stems seemed to be perfect (angles), hence not natural", which implies that "natural" is an important factor and the angles of the stems is a factor that affects the "natural" factor. There was only one participant who suggested the opposite, i.e. that artificial-looking plants looked good, or more specifically "interesting [...] as I do not expect to see a lot of symmetry in plants".

"more leaves" was the second most occurring category (5 comments). Comments that fit into this category either suggest that the plants with more leaves look better, or that plants with less leaves look worse. There were also comments that said that a balanced amount of leaves is positive (2 comments).

The third most occurring category is "detail", which says that a higher level of detail in the plants is a positive factor. Simple or small plants were often commented as having too little detail and thus not looking natural. There are three categories that are specifically for these types of plants: straight, dynamic and arrangement. They suggest that simple plants look better when growing straight upwards, being more dynamic or well arranged.

The factors were grouped into four groups: abstract, structure, foliage and other. The abstract group contains factors that are not quantifiable and thus more abstract in nature. For example it is not straight forward to quantify how "natural" a plant looks. Structure contains factors that consider the structure of the plant, i.e. the features of the branches and their patterns. Foliage contains factors that consider the leaves on the plant and their individual properties or the foliage as a whole. Other are factors that do not fit in the other groups. The grouping of the factors is shown in Table 14.

Category	Description	Occurrences
natural	Looking “natural” is positive	8
artificial	Looking “artificial” is positive	1
healthy	Looking “healthy” is positive	2
details	Higher level of detail is positive	4
environment	Environmental effects on plants is positive	3
chaos	Chaos is negative	5
symmetry	Symmetry is positive	1
asymmetry	Symmetry is negative	1
patterns	Recognizable patterns is positive	1
more leaves	More leaves is positive	5
balanced leaves	Balanced amount of leaves is positive	2
lush	Higher leaf density is positive	2
leaf pitch	Leaves pointing up is positive	1
even spread	Branches and leaves spread out evenly is positive	3
small	Small plant size is negative	3
curves	More and soft curves is positive	3
upwards	Growing upwards is positive	3
sideways	Growing sideways is positive	1
complexity	Complex branches is positive	2
clipping	Branches or leaves growing inside the ground is negative	2
obstructions	Obstructions is positive	1
straight	When plants have few leaves they look better if they grow straight up	1
dynamic	Simple plants are better when dynamic	1
arrangement	Simple plants are better when well arranged	1
visible branches	Dense amounts of visible branches is negative	1
balanced	Center of mass close to trunk is positive	1
uniform bend	A uniform bend of the branches is positive	1

Table 12: Directional factors

Category	**Occurrences**
leaf amount	3
leaf distribution	2
leaf density	1
leaf size	1
form	3
symmetry	1
growth	1
branch amount	1
branch placement	1
character	1

Table 13: Directionless factors

Abstract	Structure	Foliage	Other
natural, artificial, healthy, character, chaos, dynamic, growth, arrangement	symmetry, asymmetry, patterns, even spread, form, small, curves, upwards, sideways, complexity, visible branches, balanced, uniform bend, branch amount, straight, branch placement	more leaves, balanced leaves, lush, leaf pitch, leaf amount, leaf distribution, leaf density, leaf size	clipping, obstructions, details, environment

Table 14: Grouping of factors

6 Discussion

6.1 Grammatical Evolution

As evident from the results, the DGEL GE process performs significantly better than the brute-force approach, and the statistically significant difference in their scores support that DGEL GE finds plants with better fitness scores than brute-force does (**H1**). Additionally, the fact that GE was five times faster than brute-force means that even if both methods produce just as good individuals, GE will be faster at doing so.

The difference in duration is an interesting case because both GE and brute-force generate the same amount of individuals, and while brute-force only fills a vector of random numbers, GE must do both a crossover and mutation which are more complex functions. It is likely that this is caused by the tournament sampling and the fact that GE improves the population with each generation, reducing the complexity of the L-systems, thus reducing the time spent evaluating them. For each individual that should be in the population of the next generation, tournament selection picks a pair of individuals (with a tournament size of 2) randomly from the current population, evaluates them and allows the best one to become part of the next generation. Because of this, tournament sampling may pick the same individuals multiple times, which means the cached evaluation is used, and some individuals may not be picked at all. The fitness evaluation of a chromosome is expensive because it has to generate the L-system, interpret it into a 3D structure and then run each metric on this structure, so this may have a big impact.

6.2 Simulated Annealing

The clustering of the movements around the current score in the SA process (Figure 13) indicates that the mutation function used has a good locality, i.e. that the neighboring grammar distributions are fairly close in terms of score. At the same time, the somewhat uniform spread of scores between the current score and 0 indicates that there are some mutations that have bad locality. Additionally the clustering at exactly zero indicates that there are some mutations with the worst possible locality. These bad locality mutations could be caused by some parameters in the grammar distribution having a cascading effect. For example if the `symbol / stack` distribution in depth 0 suddenly changes from 0.5/0.5 to 1.0/0.0, all depths below will be excluded and thus most of the parameters in the distribution will

be irrelevant. The clustering at exactly 0 is more likely to be caused by the fitness measure, because it has a limit on the amount of L-system instructions and when that limit is reached the plant is in all cases scored 0. It could also happen if the distribution does not allow for any `F` symbols as without it no branches will be drawn in the 3D model.

Because brute-force with the SA-optimized distribution has a higher percentage of non-zero fitness scores than with a uniform distribution, one of the benefits of using an optimized distribution is that it avoids many of the problematic "nothing" plants that are scored 0. This is good because even though the non-zero scores in majorly zero-scored grammar distributions may have good scores (as evident from Figure 15b), a large amount of zero-scored plants will lead to a slow search process.

Because of the large amounts of zero-scored plants, as evident from Figure 16 and 15a, the median should be a better measure for central tendency. Based on this, it may be argued that the median should be used instead of the mean when measuring the grammar distributions as well. But since a large amount of zero-scored plants is bad, which is something the grammar distribution should consider, the mean could be argued as being a better measurement for the grammar distribution. Additionally, by using the median, it will suddenly jump from a good score to zero if the amount of zero-scored plants goes below 50%. Thus comparing distributions that are at around this limit by the median may be unfair, so the mean is more appropriate.

Based on the observation that the SA-optimized distribution restricts itself to three depths, setting a hard limit of maximum four depths may not have been too limiting. At the same time, other SA-optimized distributions may want to allow four depths, and there may be a point above four depths where very good plants are found.

Comparing the SA-optimized grammar distribution in Figure 20 with the generated 3D model in Figure 21, similarities can be drawn, thus indicating that the grammar distribution has an effect on the resulting plants. For example, the fact that depth 1 only has variables, and deeper depths barely has any variables, is reflected in the tall and straight tree with few branches. It is impossible for the generator to produce L-systems with operators in depth 0, and thus only straight lines are possible in that depth. While there is a possibility of drawing angled lines through the lower depths, because of the low rate of variables in them, it is less likely. The only reason that there are branches coming out of the trunk is of the string "`[&&&O>&>&&>>&&F]`" in rule `P`. This string pitches down and rolls before drawing a line so that the branch will actually stick out from the trunk rather than grow inside it.

The strong weight of F in the grammar distribution indicates that the SA process understood the importance of F. Without the F variable, the L-systems would have no branches and thus get score 0. Additionally the fact that only a small amount of the other variables are strongly weighted could be explained by the fact that a large amount of variables decreases the likeliness of the same variable appearing multiple times and therefore not being useful (e.g. only in successors).

With a significant difference in the medians by a large amount, it is clear that "an SA-optimized grammar distribution can find better individuals than a uniform grammar distribution" (**H2**) The additional fact that the SA-optimized grammar distribution produced a significantly smaller proportion of zero-scored individuals further supports this hypothesis.

As there was no statistically significant difference between GE with a uniform grammar distribution and random brute-force with an SA-optimized grammar distribution, hypothesis **H3** that random brute-force with an SA-optimized distribution can perform as well as GE with a uniform grammar distribution is supported. This is rather interesting, because this indicates that only pre-optimizing the grammar distribution can have a significant beneficial impact on the search, at least as strong impact as GE has. Though it should be kept in mind that, as discussed in Section 6.1, GE uses significantly shorter time to perform the search than a random brute-force search on the same amount of individuals, which may mean that even if random brute-force with an SA-optimized distribution can perform as well as GE with a uniform grammar distribution, the latter may be faster.

The statistically significant difference in the medians of GE with an SA-optimized distribution and GE with a uniform distribution supports hypothesis **H4** that "GE with an SA-optimized grammar distribution can find better individuals than with a uniform grammar distribution". This indicates that a grammar distribution optimized by SA not only can improve random brute-force search, but can also improve GE, and possible other EAs, to perform better with stricter parameters, thus decreasing its duration.

6.3 Fitness

Excluding participants that had null or invalid reasons, only 16% of the participants (6 people) did not strongly agree with the AHP ranking, and all of those still agreed with the ranking. Thus, there is a strong support that the AHP pairwise comparison ranking method is valid. Still there are issues with it according to the participants. 5 (83%) of them think that the rank of one plant is wrong, while 1 (17%) thinks that the distances between some ranks are wrong. This could either be an effect of the participants not being experts in the field or indicate an issue with doing pairwise comparison where they do not see the whole picture.

The designed or adapted metrics used in the fitness function were clearly not perfect measures for how aesthetically pleasing plants are. Still, as both humans and the fitness function both agreed on the two best plants and three worst plants, the metric may have the baseline of how to measure the "pleasingness" of plants. It is only in the middle region where the fitness metrics get confused and disagrees with the humans. With Kendall's tau indicating no correlation in the middle region, it is a strong indication that this region has the biggest potential for improvement. Interestingly, dRank indicates that the distance between the rankings the middle region (5.21) is smaller than the complete ranks (16.67). This effect could be caused by the reduction of the number of systems (plants), which the dRank distance is highly dependent on [48]. In any case, the p-value of dRank is still 0, indicating that they are still not correlated. Based on this, the hypothesis that humans agree with the ranking made by the DGEL fitness component (**H5**) is rejected, and the next question should be why they do not agree and what could be done to improve the agreement.

The categorization of factors that humans find important can give an overview of what the fitness function should include or exclude. Directional factors are directly useful since they can be used to change a fitness metric's direction or create new metrics for the fitness function. Directionless factors are less useful on their own, but they indicate the importance of the factor, and thus can either be used to adjust the weight of a metric or a directional factor. For example, "leaf amount" is a directionless factor that can be used to strengthen the directional factors "more leaves" and "balanced leaves". Both types of factors can be used to analyze why the current metrics do not correlate with the human ranking.

With the standard errors being smaller on the bad plants than on the good plants, it might indicate that it is more likely that humans agree on what looks bad compared to what looks good. This is also supported by the observations of the four first participants where they often spent more time comparing good plants to each other, and said themselves that this was difficult.

Drop, closeness and branching were found as the metrics that had the worst correlations, and the correlation statistic on the rankings with these metrics removed supports this. It is clear that these metrics either incorrectly measure pleasingness or measure something else.

Closeness is most likely not a good metric for how aesthetically pleasing a plant is, because its purpose is to prevent a the physically impossible phenomenon of branches clipping into each other, something that the human in most cases does not notice. It could in fact have a negative effect on the fitness function, because one of its artifacts is that multiple leaves will appear on the same branch segment when multiple branches clip into each other. This artifact was indicated by one

participant in the survey as “creating an interesting symmetry” that they liked. A possible solution could be to remove this metric and use a heuristic or physical simulation to make the branches collide with each other and thus bend away from each other. Other metrics, such as *branching*, could then reward or punish the branching point like in other cases. The artifact of multiple leaves at the same segment could be added as a heuristic, or it would be allowed with an L-system grammar such as the one in Listing 2.

The drop metric is a similar type of metric in that it also aims to prevent physically impossible situations. It was originally designed to prevent plants from growing down through the ground, but as the plants viewed in the survey were placed on a pyramid-shaped hill, this was not an issue. Thus the reason it had a bad correlation is most likely similar to that of the closeness metric. This could also be solved with a heuristic or physical simulation, like with the other metric.

The branching metric is a more interesting case as it is more aimed at measuring the realism of the plant, but still has a bad effect on the overall fitness function. From the plot in Figure 30, it seems to work well in distinguishing the good from the bad plants, but it creates a bump in the middle region from rank 5 to 8. Looking at the specific plants in this region, there could be multiple reasons for this bump. The plant at rank 5, when counting the branches coming out from the bases, looks like it has an OK amount of branches, but is generally simplistic and small, which are negative factors that could be overshadowing the other factors such as the branching. The next plant has a balanced amount of branches, but that value may have been overshadowed by its simplicity, leaf scarcity and symmetry which are all mostly negative factors as indicated by the humans. The plant at rank 7, when looking at it, has a large amount of branches coming out of a single point, indicating that either the metric is not correctly measuring the branching or the branches are clipping into each other, thus allowing more branches to appear from the same point. The latter is a likely cause because the closeness metric is punishing it with the maximum possible value (-1.0), meaning that there is at least one point where branches completely overlap. The last plant visually has slightly too many branches, which is also supported by the *branching* metric (0.38, where the best is 1 and worst is -1), but is not on the negative side. The humans also rated this plant the highest of all four middle plants. Compared to the plants ranked above it, it looks smaller, simpler and less lush, which may again overshadow the branching.

Based on the above analysis of the middle plants, it could be argued that the size, complexity and lush factors should be included in the fitness or more heavily weighted. None of them are directly included in the current fitness metric, but *length*, *branching* and *foliage* should cover parts of them. Additional metrics could include something that measures the spread of the plant in addition to its length

(size factor), density of branches (complexity) and density of leaves (lush). They should prefer bigger, denser and more lush plants, but they should also have a balance where at some point it becomes too much. Additionally, because three of these plants were relatively small, and humans indicated that when this is the case their level of detail is too low, metrics measuring the size and complexity factors could dynamically be weighted more based on their score, overshadowing the other metrics if necessary.

6.4 Research Questions

As parametric S2L-systems are the most flexible L-system variations, one can argue that they should be the best choice for L-system plant models that "are appropriate to represent plants both found in nature and not found in nature" (**RQ1**). While these L-systems were not used in this research and other literature because of their complexity, based on the results, simple D0L-systems, specifically represented by the grammar in Listing 1, seem to be able to model both natural and artificial plants. This is because, based on the comments from the participants, both "natural" and "artificial" was found as important factors either describing bad or good plants, thus indicating that the generated plants was able to look both natural and artificial. An important factor to consider is that the L-system grammar only enabled branches, and adds branch widths and leaves as simple heuristics. Thus, these results are only applicable if these heuristics or L-system grammar that replace these heuristics are used. While D0L-systems may be appropriate to represent both natural and artificial plants, it may not be the best. More flexible models, such as IL-systems, parametric L-systems, stochastic L-systems, or their combination as parametric S2L-systems may be better as they allow for more detail.

That the GE performance improved with increasing crossover rates indicates that the crossover operation has a positive effect on the evolution. This could suggest that the DGEL chromosome representation could be a good choice for "[combining different L-systems] into new plants" (**RQ1**). It does not directly indicate that one single crossover will retain or improve the fitness of the plant, but using it in a GE process either targeting a good fitness or the same fitness as the parents could work. For example, the initial population could consist of 50% of each parent, the mutation rate could be set to 0.1 and crossover rate to 0.5. These recombination rates focuses more on crossover than mutation, thus being more likely to keep some of the features in the chromosome, but are still able to reach a fitness of 0.92. Alternatively, since the DGEL generation process is able to produce fit individuals, it could be used together with a fitness measure of how similar the offspring is to its parents.

While the fitness function used does not correctly measure how aesthetically

pleasing an L-system plant is, based on the analysis, it has the potential for it. Because the rankings done by humans only indicate the relative differences between plants and not how pleasing a plant is on an absolute scale, it is difficult to say if DGEL manages to generate aesthetically pleasing L-system plants (**RQ2**). What we can say is that DGEL is able to generate both plants that are less aesthetically pleasing and more aesthetically pleasing.

To know if the "more aesthetically pleasing" plants are actually aesthetically pleasing, they would have to be evaluated in a different manner. For example, the plants from this survey could be used in another survey where participants are asked if they find the presented plants aesthetically pleasing or not. This could also indicate a threshold in the fitness function which must be reached for a plant to be pleasing. This assumes that the fitness function and humans agree on the ranking.

Looking at the observations of participants in the survey, some said that certain plants looked realistic or natural. Based on this and the assumption that realistic or natural plants are aesthetically pleasing, this could suggest that DGEL is able to generate aesthetically pleasing L-system plants (**RQ2**), though this is a stretch.

With the in-depth analysis of one SA-optimized grammar distribution, there is an indication that the DGEL grammar distribution together with SA can focus the parameter space onto certain areas that contain plants with particular properties. It shows that the grammar distribution can restrict the stack dept, adjust in which depth the drawing instruction should appear and control which rotations should be more prominent. Thus, by using multiple grammar distributions, multiple "generators" that each generate somewhat unique plants could be used together to create a larger variations of plants.

Additionally, the 37 different factors of aesthetic plants found from the qualitative data from participants, indicate that just the 12 different plants generated have different properties and can therefore be varied. For example, both natural, artificial, chaos, symmetry, asymmetry, curves, straight, sideways, environment and details were found as factors. They are all different or opposites of others, and since the participants were asked specifically about the plants that they ranked, this shows that there was a clear variation in those 12 plants. The fact that these 12 plants were randomly generated by DGEL suggests that this may apply generally for DGEL and that another 12 new plants may also be varied. Thus, using L-systems with GE, and optionally using grammar distributions with parameter optimizers is a method of generating varied L-system plants (**RQ3**). Some hand-picked plants that were generated with DGEL can be found in Appendix B.

All of this together suggests that the DGEL architecture and algorithm is a viable solution to generating L-system plants "that are aesthetically pleasing, varied and could be used to create offspring similar to its parents" (**RQ0**). With the additional

support of hypotheses **H2**, **H3** and **H4**, DGEL (with help from SA) is shown to be more efficient than plain GE, and the fact that an SA-optimized grammar distribution to a high degree helps to avoid zero-scored plants, indicates that it, and thus DGEL as a whole, better handles complex parameter spaces.

6.5 Implications of Findings

The concept of DGEL is not restricted to L-systems. By removing the L in DGEL, thus becoming Distribution-based Grammatical Evolution (DGE), the same methods may be applicable to other grammar-based systems. Particularly the novel approach of using a grammar distribution for GE is a method that other problems may benefit from.

Clearly, based on the tests of an SA-optimized distribution's performance (**H2**, **H3** and **H4**), using a grammar distribution is a significant benefit. Particularly interesting is to see the optimized grammar distribution applied to random brute-force perform as well as GE, and possibly even better if there was more leeway in the score (there was no room for improvement). Combined with the indication that the grammar distribution improves the performance of GE, it shows that a grammar distribution can be a significantly useful tool in the evolutionary computing toolbox.

First of all this means that L-systems can be generated more efficiently, and more complex L-systems can be generated, resulting into more, better and varied 3D plants, something PCG and thus the game industry and consumers can benefit of. Secondly, DGEL applied to other grammar-based problems can be just as useful. While only the generation of L-systems was studied, the results (except for the aesthetics) will most likely be reflected in other grammar-based problems. This means that a complete series of grammar-based problems can also benefit from using optimized grammar distributions. Again this is something PCG, and thus the game industry and consumers, can benefit of because grammars can be used to generate other game content such as missions, spaces and levels [22]. Another good example of its benefit is in the use of grammars to generate and evolve computer programs [28].

In more general or abstract terms, the grammar distribution is a probability distribution. Therefore our findings suggest that accompanying an EA with a probability distribution, and optimizing it using optimization techniques, can improve its efficiency in complex spaces, allowing for faster searches and more varied solutions.

6.6 Limitations

There are limitations to the findings, especially regarding the variations in plants and the fitness evaluation. Only one SA-optimized grammar distribution was studied, and only one plant generated with that grammar distribution. While this gave an in-depth view on this part of DGEL, the external validity of the results is questionable. It is possible that SA finds similar grammar distributions each time and thus does increase the variation of the plants. The variation within and between different grammar distributions (including both optimized and uniform) should be studied to see if this is generally true or not.

There is a big limitation in the sampling method used and the sample of humans found for the evaluation of the fitness metric. As seen from the overview of the demographics, the sample is highly biased and thus does most likely not represent the population.

Another limitation is that the GE parameters found may not be the optimal parameters. For example, the tournament size did not have an effect on the performance of the GE process, which may be caused by the large population size and number of generations already being good enough. It might be the case that a larger tournament size may lead to better performance with smaller population size and fewer generations. In fact, tournament has an increasing selection intensity with increasing tournament size [34], meaning that the fitness should change faster with larger tournament sizes. The same problem may apply to the other parameters as well. This limitation does not affect the rest of the results, it only means that the GE parameters could likely be improved.

As heuristics are used for both branch widths and leaf placement, the results may not apply to L-systems that include leaves and widths as parts of the grammar. A grammar like that would have more dimensions and therefore be more complex to search, thus the results may be different. The results could both be worse or better. For example the benefit of using GE or SA may be even larger when the space is more complex, or the space may be too complex for them to perform well. Especially the particular parameters used, for example the GE parameters, may not be applicable for other L-system grammars. Additionally, using other L-systems, like parametric SIL-systems introduce even more dimensions to the search space, which may have the same problem.

Finally each person only saw one plant alone, which is unusual in nature. Realistically plants would be surrounded by other plants, either of the same species or of different species, which will likely have a significant effect on the perception of the plant. The perception of the plant may also be different depending on the context it is in. For example a potted plant in the window inside a house may have different criteria for being aesthetically pleasing than a plant in the garden, or a

plant found in the wild.

7 Conclusion

This thesis explored how to improve plant L-system generation, particularly with the goal of being usable for combining plants. The idea behind this is to be able to create a virtual world where people may grow and cross-breed plants. For people to enjoy this world, the plants should be aesthetically pleasing. They should also be varied so that the people may get new experiences. Finally the plants should be modeled in such a way that they can be combined into new plants via cross-breeding. Thus, the research question is formed: How can we generate plants that are aesthetically pleasing, varied and could be used to create offspring similar to its parents?

Previous research on L-system generation has been focused on restricted L-system grammars and with fitness functions that aim at generating realistic plants. Multiple L-system versions have been used, including D0L-systems, PD0L-systems, and PDIL-systems, which have then been evolved using evolutionary algorithms including GA, GP and GE. Both autonomous fitness evaluation of L-systems, human evaluation and similarity evaluation have been used as selections strategies in the evolutionary algorithms. Several different metrics have been used to measure the fitness of a plant, mostly aiming at evaluating the survivability of the plant, though some use humans to aesthetically evolve the plants. Multiple genetic operators have been used to modify the the L-system genes in order to evolve them, and they are often specifically designed for L-systems in order to not invalidate the syntax, and often only parts of the L-systems can be modified.

We introduced a concept called Distribution-based Grammatical Evolution of L-systems (DGEL), based on the use of GE to evolve L-systems [28, 27, 11]. It follows the traditional GE method [28], but introduces a grammar distribution to control which parts of the grammar that should be weighted the most. Because varied plants are required, DGEL does not restrict the grammar and modification of the L-system as much as the previous research does. But with a less restricting grammar, the search space becomes significantly larger. Thus the distribution will help focus the search space without actually restricting it. Additionally, different distributions may produce particular plants and thus multiple distributions could be used together to generated varied plants.

At the heart of DGEL is the model of the L-system, which consists of a chromosome, a grammar description and a grammar distribution. This novel underlying representation of the L-system adapts GE to work with a distribution, and is also

applicable to problems not involving L-systems, such as evolution of programs defined by programming languages. DGEL consists of two processes that generate parts of the model. SA is used to find grammar distributions that GE will used to evolve chromosomes that become aesthetically pleasing plants. To measure how aesthetically pleasing the plants are, metrics are adapted from the literature and some new are created. They focus on the structure and shape of the plant, the foliage on it, and physically impossible situations. To simplify the problem, only branches were allowed in the L-systems, and branch widths and leaves were added as heuristics to the generated plant so that they would not look dead.

DGEL was evaluated in three parts: GE, SA/distribution and fitness. Each of GE's parameters, except for the tournament size, were found to be improving the performance of it with larger values. The crossover operator rate interestingly improved the performance until it went from 0.5 to 1.0 where it worsened the performance by a large margin. GE was shown to be an improvement over random brute-force search when both generated the same amount of individuals. It was additionally several times faster than random brute-force search, likely caused by the tournament sampling not evaluating each individual.

The SA's progress during one run was analyzed and it was found that it was able to find distributions that on average generated plants with a fitness of 0.6, compared to 0.0 for the uniform distribution. It had mostly good locality, but some moves had the worst possible locality, likely caused by cascading effects of some of the parameters. The SA-optimized grammar distribution was found to generate plants with higher scores than a uniform distribution, and a random brute-force search method with the SA-optimized grammar distribution could in fact find better plants than GE with a uniform grammar distribution. GE was additionally found to perform better when an SA-optimized grammar distribution is used instead of a uniform grammar distribution.

The grammar distribution and an L-system generated by the SA-optimized grammar distribution was studied in-depth to see what effect it could have. It was found that the grammar distribution had a restricting effect on the L-system, making the produced plants possibly have similarities with each other, which suggests that multiple distributions could be used together to generate more varied plants. Additionally, based on feedback from people evaluating the generated plants, 37 factors were found as important for distinguishing the good plants from the bad, indicating that DGEL was able to generate a wide specter of plants, even without an optimized distribution.

A survey was created where participants ranked 12 generated plants using pairwise comparisons and AHP. It was found that humans did not perfectly agree with the ranking created by the fitness function, though Kendall's tau indicated a pos-

itive correlation. There was a general agreement on which plants that were the best and the worst, but the middle part had no correlation. The metrics *branching*, *drop* and *closeness* were found to have the worst correlation, which affected the correlation between the fitness ranking and the human ranking. The cause is most likely that two of them are not measuring the aesthetics of the plant, and the last one is overshadowed by other more important factors in the humans' minds. A suggested solution to this is to include additional metrics based on the human feedback that include size, branch density and leaf density. Additionally, size and complexity should be weighted more when they have worse scores such that they overshadow the branching metric for small and simple plants.

All of this together suggests that the presented DGEL is a solution to generating plants "that are aesthetically pleasing, varied and could be used to create offspring similar to its parents". By removing the L in DGEL, it could be used for other problems that are not related to plants or L-systems, but use grammars to describe a system, for example computer programs. Furthermore, by looking at in more abstract terms, the findings suggest that accompanying an EA with a probability distribution, and optimizing it using optimization techniques, can improve its efficiency in complex spaces, allowing for faster searches and more varied solutions.

Though there are limitations to the project, especially concerning the survey and the analysis of the SA process. The sample used in the survey is highly biased making the results not directly generalizable to the population, but it still gives an indication of what may be the truth. The study of the SA process only studies one SA run, one grammar distribution and one produced plant, meaning that the findings may not be applicable to other SA runs, other distributions and other plants. At the same time, it is an in-depth analysis that suggests that the SA process "knows what it is doing", and similarities drawn between the grammar distribution and the produced L-systems and plant suggests that it can have an important effect.

7.1 Future Work

DGEL has weaknesses that need to be resolved to make it better. Especially the fitness function needs to be improved as it is the most central part of generating "aesthetically pleasing plants". A larger study with a representative sample should be conducted to understand what metrics are important for measuring aesthetically pleasing plants. Physical simulation should be tested as an alternative to the *closeness* and *drop*, though this will make the fitness dependent on the environment. How the context around the plant affects how aesthetically pleasing it is should also be studied.

The plant similarities within and between SA-optimized grammar distributions

should be studied to find if they are focusing on different areas in the parameter space, or if all are focusing on the same space. This would require a metric for the similarities between L-system plants, which is also something DGEL could use to cross different L-systems in order to produce offspring that are similar to its parents.

It is also necessary to study how DGEL handles more complex grammars, such as parametric and context-sensitive L-systems, or simply D0L-systems with more features like leaves, branch widths and colors implemented in the grammar instead as heuristics. With more flexible L-systems more dynamics in the plants can be created, and thus DGEL should be able to generate more interesting and varied plants.

Another interesting problem for the future is how chromosomes from different grammar distributions can be combined. This would be important if DGEL were to be used for combining plants of large varieties into offspring. The chromosome and grammar distribution are strongly tied, and can therefore not be simply combined. A possible solution that should be explored is to convert parent chromosomes from non-uniform grammar distributions to uniform distributions after they have been generated, thus enabling them to be combined assuming a uniform grammar distribution. Would the optimized distributions then only be useful for generating the initial population of plants, or could they be used inside the grammatical evolution process by converting forth and back between distributions? Additionally, could this conversion be made lossless, enabling a converted chromosome to be converted back to the exact original chromosome?

Beaumont and Stepney find indications that both elitism in the GA selection and structure in the grammar affects the performance of GE [11]. The changes in the grammar are subtle and does not change the possible solutions, but still have a significant effect on the performance in some cases. Thus this may apply to DGEL as well and should be investigated. It is especially interesting if this effect is mitigated by using a grammar distribution.

Bibliography

[1] Prusinkiewicz, P. & Lindenmayer, A. 1990. *The algorithmic beauty of plants*. Springer-Verlag, Electronic version: 2012 (Accessed: November 13, 2016). URL: http://algorithmicbotany.org/papers/#abop.

[2] Lindenmayer, A. 1968. Mathematical models for cellular interactions in development I. filaments with one-sided inputs. *Journal of Theoretical Biology*, 18(3), 280 – 299. doi:10.1016/0022-5193(68)90079-9.

[3] Lindenmayer, A. 1968. Mathematical models for cellular interactions in development II. simple and branching filaments with two-sided inputs. *Journal of Theoretical Biology*, 18(3), 300 – 315. doi:10.1016/0022-5193(68)90080-5.

[4] McCormack, J. 1993. Interactive evolution of l-system grammars for computer graphics modelling. *Complex Systems: from biology to computation*, 118–130.

[5] Ashlock, D. 2006. Evolutionary computation for modeling and optimization.

[6] Mock, K. J. May 1998. Wildwood: the evolution of l-system plants for virtual environments. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, 476–480. doi:10.1109/ICEC.1998.699854.

[7] Ochoa, G. *On genetic algorithms and lindenmayer systems*, 335–344. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. doi:10.1007/BFb0056876.

[8] Ebner, M., Grigore, A., Heffner, A., & Albert, J. *Coevolution Produces an Arms Race among Virtual Plants*, 316–325. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002. doi:10.1007/3-540-45984-7_31.

[9] Ebner, M. *Evolution and Growth of Virtual Plants*, 228–237. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. doi:10.1007/978-3-540-39432-7_25.

[10] Ashlock, D., Bryden, K. M., & Gent, S. P. 2006. Simultaneous evolution of bracketed l-system rules and interpretation. In *2006 IEEE International Conference on Evolutionary Computation*, 2050–2057. doi:10.1109/CEC.2006.1688559.

[11] Beaumont, D. & Stepney, S. May 2009. Grammatical evolution of l-systems. In *2009 IEEE Congress on Evolutionary Computation*, 2446–2453. doi:10.1109/CEC.2009.4983247.

[12] Corchado, M. A. R., Ruiz, J. C. S., Corchado, F. F. R., & Romero, J. R. M. Jan 2009. Growing plants for virtual 3d environments. In *2009 6th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, 1–6. doi:10.1109/ICEEE.2009.5393469.

[13] Jacob, C. *Genetic L-System Programming*, 333–343. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994. doi:10.1007/3-540-58484-6_277.

[14] Vanak, S. K. 2000. X-machines+l-systems=xl-systems. In *Proceedings Computer Graphics International 2000*, 127–134. doi:10.1109/CGI.2000.852328.

[15] Hornby, G. S. & Pollack, J. B. 2001. Evolving l-systems to generate virtual creatures. *Computers & Graphics*, 25(6), 1041 – 1048. Artificial Life. doi:10.1016/S0097-8493(01)00157-1.

[16] Jacob, C. 1995. Genetic l-system programming: breeding and evolving artificial flowers with mathematica. In *Proceedings of the First International Mathematica Symposium*, 215–222.

[17] Jacob, C. *Evolution programs evolved*, 42–51. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996. doi:10.1007/3-540-61723-X_968.

[18] Jacob, C. 1996. Evolving evolution programs: Genetic programming and l-systems. In *Proceedings of the 1st Annual Conference on Genetic Programming*, 107–115, Cambridge, MA, USA. MIT Press. URL: http://dl.acm.org/citation.cfm?id=1595536.1595550.

[19] Piperno, D. R., Ranere, A. J., Holst, I., Iriarte, J., & Dickau, R. 2009. Starch grain and phytolith evidence for early ninth millennium bp maize from the central balsas river valley, mexico. *Proceedings of the National Academy of Sciences*, 106(13), 5019–5024. doi:10.1073/pnas.0812525106.

[20] Acquaah, G. 2009. History and role of plant breeding in society. In *Principles of plant genetics and breeding*, chapter 1. John Wiley & Sons.

[21] Hartung, F. & Schiemann, J. 2014. Precise plant breeding using new genome editing techniques: opportunities, safety and regulation in the eu. *The Plant Journal*, 78(5), 742–752. doi:10.1111/tpj.12413.

[22] Togelius, J., Shaker, N., & Dormans, J. 2016. Grammars and l-systems with applications to vegetation and levels. In *Procedural Content Generation in Games*, chapter 5. Springer.

[23] Togelius, J., Kastbjerg, E., Schedl, D., & Yannakakis, G. N. 2011. What is procedural content generation?: Mario on the borderline. In *Proceedings of the 2Nd International Workshop on Procedural Content Generation in Games*, PCGames '11, 3:1–3:6, New York, NY, USA. ACM. doi:10.1145/2000919.2000922.

[24] Julian Togelius, N. S. & Nelson, M. J. 2016. Introduction. In *Procedural Content Generation in Games*, chapter 1. Springer.

[25] Koza, J. R. 1992. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press.

[26] O'Neil, M. & Ryan, C. *Grammatical Evolution*, 33–47. Springer US, Boston, MA, 2003. doi:10.1007/978-1-4615-0447-4_4.

[27] Ortega, A., Dalhoum, A. A., & Alfonseca, M. July 2003. Grammatical evolution to design fractal curves with a given dimension. *IBM Journal of Research and Development*, 47(4), 483–493. doi:10.1147/rd.474.0483.

[28] Ryan, C., Collins, J., & Neill, M. O. *Grammatical evolution: Evolving programs for an arbitrary language*, 83–96. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. doi:10.1007/BFb0055930.

[29] Ding, W., Hu, C., & Zhu, Y. *Simulating Virtual Plants Based on Genetic Algorithm and L-Systems*, 145–153. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. doi:10.1007/978-3-642-38466-0_17.

[30] McCormack, J. *Aesthetic Evolution of L-Systems Revisited*, 477–488. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. doi:10.1007/978-3-540-24653-4_49.

[31] Crocker, D. & Overell, P. Augmented bnf for syntax specifications: Abnf. STD 68, RFC Editor, January 2008. http://www.rfc-editor.org/rfc/rfc5234.txt. URL: http://www.rfc-editor.org/rfc/rfc5234.txt.

[32] Özdamar, L. & Demirhan, M. 2000. Experiments with new stochastic global optimization search techniques. *Computers & Operations Research*, 27(9), 841 – 865. doi:10.1016/S0305-0548(99)00054-4.

[33] Onbaşoğlu, E. & Özdamar, L. Jan 2001. Parallel simulated annealing algorithms in global optimization. *Journal of Global Optimization*, 19(1), 27–50. doi:10.1023/A:1008350810199.

[34] Blickle, T. & Thiele, L. A comparison of selection schemes used in genetic algorithms. Technical report, GLORIASTRASSE 35, CH-8092 ZURICH: SWISS FEDERAL INSTITUTE OF TECHNOLOGY (ETH) ZURICH, COMPUTER ENGINEERING AND COMMUNICATIONS NETWORKS LAB (TIK), 1995.

[35] Razali, N. M., Geraghty, J., et al. 2011. Genetic algorithm performance with different selection strategies in solving tsp. In *Proceedings of the world congress on engineering*, volume 2, 1134–1139.

[36] The rust programming language (online). URL: https://www.rust-lang.org/en-US/ (Visited 12.12.2017).

[37] Frequently asked questions - the rust programming language (online). URL: https://www.rust-lang.org/en-US/faq.html (Visited 12.12.2017).

[38] Unsafe - the rust programming language (online). URL: https://doc.rust-lang.org/book/first-edition/unsafe.html (Visited 12.12.2017).

[39] Data races and race conditions - the rust programming language (online). URL: https://doc.rust-lang.org/nomicon/races.html (Visited 12.12.2017).

[40] Vik, M. B. December 2017. Gachapen/lsystem v0.1.3. doi:10.5281/zenodo.1116259.

[41] Shapiro, S. S. & Wilk, M. B. 1965. An analysis of variance test for normality (complete samples)†. *Biometrika*, 52(3-4), 591–611. doi:10.1093/biomet/52.3-4.591.

[42] Wilk, M. B. & Gnanadesikan, R. 1968. Probability plotting methods for the analysis for the analysis of data. *Biometrika*, 55(1), 1–17. doi:10.1093/biomet/55.1.1.

[43] Brown, M. B. & Forsythe, A. B. 1974. Robust tests for the equality of variances. *Journal of the American Statistical Association*, 69(346), 364–367. doi:10.1080/01621459.1974.10482955.

[44] Mann, H. B. & Whitney, D. R. 03 1947. On a test of whether one of two random variables is stochastically larger than the other. *Ann. Math. Statist.*, 18(1), 50–60. doi:10.1214/aoms/1177730491.

[45] Leedy, P. D. & Ormrod, J. E. 2014. *Pearson New International Edition: Practical Research: Planning and Design*. Pearson Education Limited, 10th edition.

[46] Saaty, T. L. 2008. Decision making with the analytic hierarchy process. *International Journal of Services Sciences*, 1(1), 83–98. doi:10.1504/IJSSci.2008.01759.

[47] KENDALL, M. G. 1938. A new measure of rank correlation. *Biometrika*, 30(1-2), 81–93. doi:10.1093/biomet/30.1-2.81.

[48] Carterette, B. 2009. On rank correlation and the distance between rankings. In *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, 436–443, New York, NY, USA. ACM. doi:10.1145/1571941.1572017.

[49] Webber, W., Moffat, A., & Zobel, J. November 2010. A similarity measure for indefinite rankings. *ACM Trans. Inf. Syst.*, 28(4), 20:1–20:38. doi:10.1145/1852102.1852106.

[50] Media formats for HTML audio and video (online). URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Supported_media_formats#Browser_compatibility (Visited 08.12.2017).

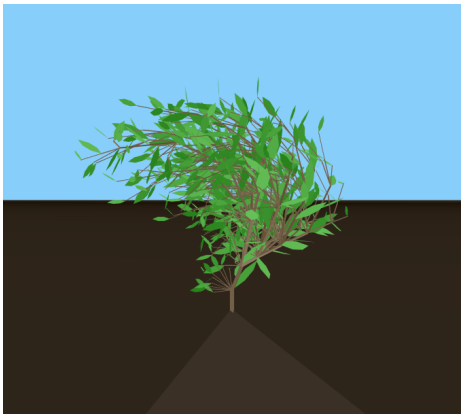
[51] FFmpeg and VP9 Encoding Guide (online). URL: https://trac.ffmpeg.org/wiki/Encode/VP9 (Visited 08.12.2017).

[52] Recommended Settings for VOD (online). URL: https://developers.google.com/media/vp9/settings/vod/ (Visited 08.12.2017).

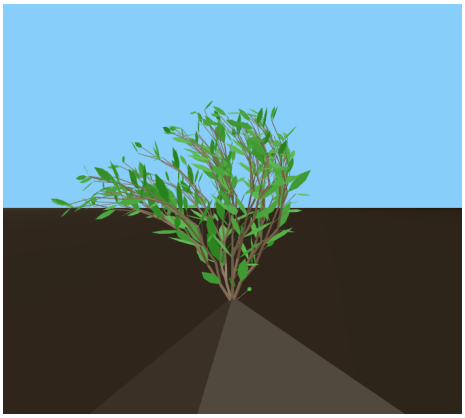
[53] H.264 Video Encoding Guide (online). URL: https://trac.ffmpeg.org/wiki/Encode/H.264 (Visited 08.12.2017).

[54] Vik, M. B. December 2017. Gachapen/lsys-pairwise v1.0.1. doi:10.5281/zenodo.1116264.

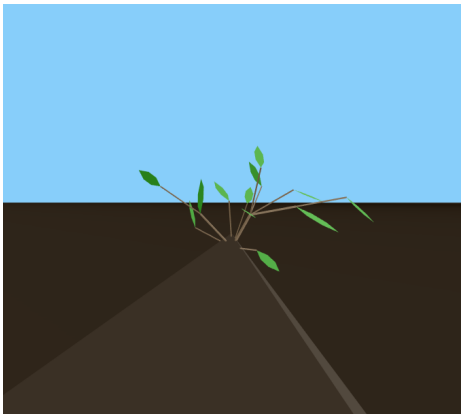
A Plants Used in Survey



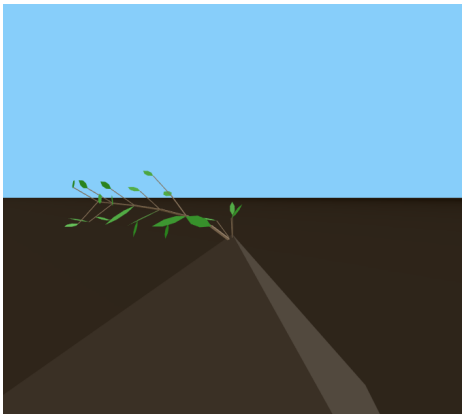
(a) Survey plant 0.97



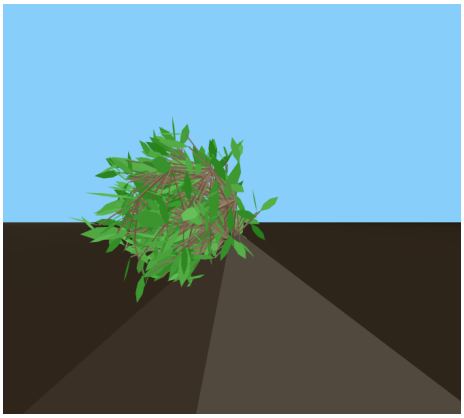
(b) Survey plant 0.91



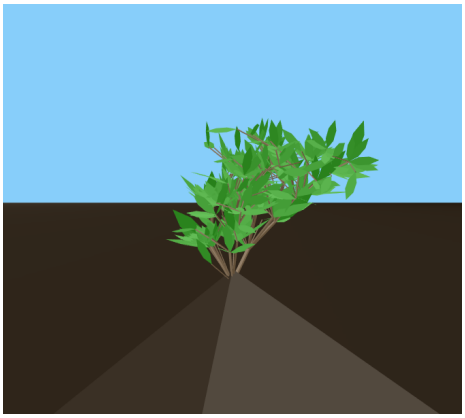
(c) Survey plant 0.84



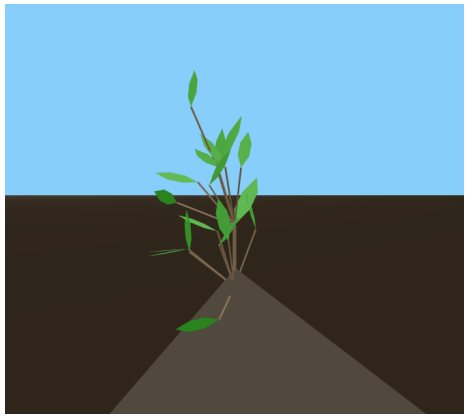
(d) Survey plant 0.78



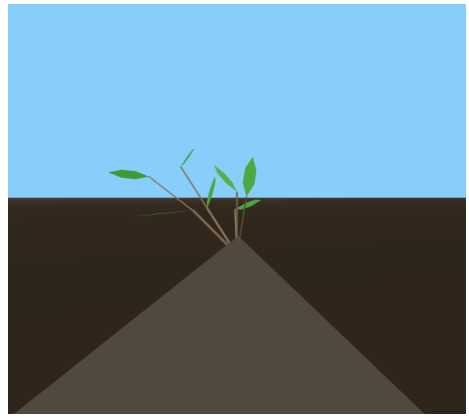
(e) Survey plant 0.72



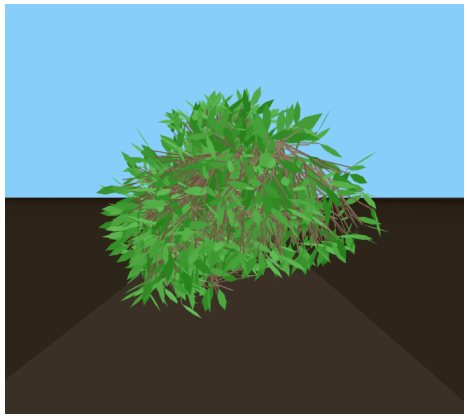
(f) Survey plant 0.65



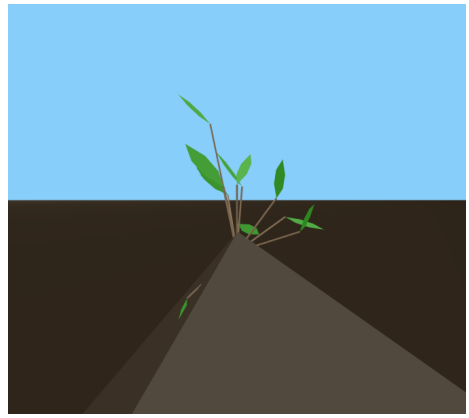
(a) Survey plant 0.59



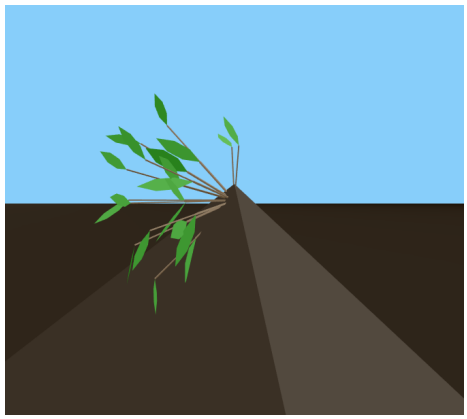
(b) Survey plant 0.53



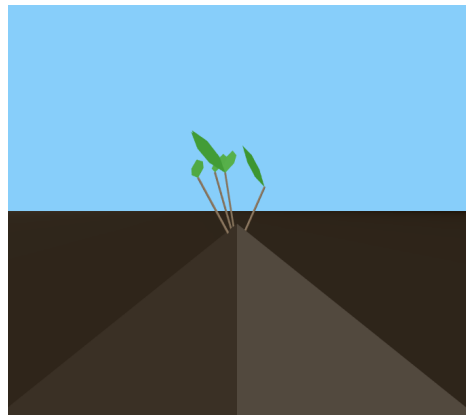
(c) Survey plant 0.46



(d) Survey plant 0.40



(e) Survey plant 0.34



(f) Survey plant 0.27

Figure 33: All plants used in the survey, in descending rank order. Their name is equal to their fitness score.

B Various DGEL Generated Plants

The next page shows a collage of various hand-picked L-system plants generated with DGEL.