NTNU – Trondheim
Norwegian University of
Science and Technology

Automatic road network generation with L-systems and genetic algorithms

Bjørn Gunnar Eilertsen

Master of Science in Computer Science
Submission date: Januar 2013
Supervisor: Pauline Haddow, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Sammendrag

Denne oppgaven er et forskningsprosjekt innunder datateknikk og kunstig intelligens, da nærmere bestemt bio-inspirert kunstig intelligens. Prosedurelle teknikker, inspirert av biologiske utviklingsmodeller, er kjent for generere komplekse strukturer fra kompakte beskrivelser. Dette motiverer utforskning av prosedurelle teknikker i bruk for å automatisk generere tredimensjonale veinettverksmodeller. I denne oppgaven er L-systemer brukt til å prosedurelt utvikle vide veinettverk, og genetiske algoritmer er brukt til å finjustere prosedyren etter gitte mål. Den endelige løsningen innbefatter også en maksimal flyt-algoritme kombinert med en genetisk algoritme for å finjustere veikryss i veinettet.

Preface

This thesis is a result of my project work for the Master's degree in Computer and Information Science at the Norwegian University of Science and Technology (NTNU). The work was carried out at the Department of Computer and Information Science (IDI) under supervision of Pauline C. Haddow.

Acknowledgements

I would like to thank my supervisor Pauline C. Haddow for all her patience, advice and guidance throughout this thesis.

I would also like to thank Jo Skjermo at the Norwegian Public Roads Administration (Statens Vegvesen) for insight and guidance on road networks.

Special thanks to my significant other, Nina Manh, for reviewing this thesis, and for all encouragement and support given through the project.

Finally I would like to thank my family for all their support through the time it took me to write this thesis.

Bjørn Gunnar Eilertsen
Oslo, January 17, 2013

Abstract

This thesis is a research in computer science and artificial intelligence, more precisely a research in biologically inspired methods. Procedural techniques, inspired by biological developmental models, are known to create complex structures from compact descriptions. This motivates exploring procedural techniques on the subject of automatically generating 3D road network models. In this thesis are L-systems applied to procedurally develop vast road networks, and genetic algorithms (GAs) are applied to tune the procedure to target outcomes. The final solution also incorporates a maximum flow algorithm combined with a genetic algorithm to tune intersections in the road network.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background and Motivation

This thesis is a research and implementation of methods that are capable of automating 3D road network generation. The methods chosen are mainly in the field of bio-inspired artificial intelligence. The motivation for automating and easing the process of generating 3D road network models is driven by the slow process of manually generating 3D models with generic modelers.

The Norwegian Public Roads Administration (NPRA) expressed a need of such automation. They are currently using a 3D studio application, but the creation time of a high detail road network is long, and it would be highly beneficial to automate this process. This motivates for a research in methods capable of automatically creating road network models with the desired detail.

This thesis is an implementation and research project of methods and theory found in a prior specialisation project, written by the author in prior semester.

1.2 Goals and Research Questions

For this project and thesis, a upper goal with underlying research questions has been defined. The goal of this research project and thesis is to apply L-systems, combined with genetic algorithms, in an attempt to automatically create 3D road network models.

Goal Automatic road network model generation by combining L-systems and genetic algorithms.

Following is a list of research questions that will be adressed:

Research Question 1 What are the minimal and required features of an L-system to create valid road network models?

Research Question 2 Can an L-system be guided by a genetic algorithm in search of better automation of road network models.

Research Question 3 What fitness measures can guide the genetic algorithm?

Research Question 4 Will a maximum flow-driven GA be able to tune flow transitions in the interpreted road network?

To achieve automation in this way, a specific L-system must be found, or L-system template. L-systems come in great variations, so a leading question is what are the essential features needed to be present in an L-system to generate valid road network models.

Road network models are complex. Procedural systems, such as L-systems, promise complex models made from small descriptions. The same developmental nature make it difficult to predict the final outcome of small input parameters. A genetic algorithm is applied in attempt to optimise these initial parameters, towards final goals. This leads to a question concerning which features of an L-system can be successfully tuned, and an additional question of which features of the interpreted L-system are helpful to steer the tuning. These two research questions concern both ends of the genetic algorithm, and both ends of the L-systems, and how these points should be linked to give the best results. While an L-system combined with a genetic algorithm may come far concerning the initial macroscopic layout of roads, setting transitions such as intersections may be better of analysing a simulation of flow through the network.

1.3 Research Method

This research implements a theoretic model and continuously conducts experiments with the implementation, to analyse and step-wise extend the implemented model. For extensibility he model is implemented as a framework of modules, which was iteratively improved throughout the project, and extended as testing gave satisfying results. A potential pitfall genetic algorithms and L-systems especially is that there are so many variations, combinations and parameters to tinker with, that the research progress demands sub-goals and milestones to be followed.

1.4 Thesis Structure

The thesis structure is as follows:

Chapter 1 gives a introduction to the thesis and problem description.

Chapter 2 describes the background theory and motivation for this project.

Chapter 3 describes the theoretical model and framework that is implemented.

Chapter 4 presents a series of experiments and discussion of results.

Chapter 5 gives an evaluation and conclusion of the work done, and describes possible future work.

Chapter 2

Background Theory and Motivation

2.1 Background Theory

This section aims to provide the background knowledge,terminology and foundation to understand the implemented model and framework. Those familiar with road network theory, L-systems, genetic algorithms and flow networks may skip ahead to Chapter 4.

2.1.1 Roads, streets and motorway networks

A road network is often best viewed on a map, different colored lines meet at crossings. On the road segment level, a road may be one of many categories. These categories vary from country to country, and apply in different contexts and settings. This thesis will focus on the Norwegian road networks, and take use of Norwegian law and guidelines in design. The Norwegian Public Road Association's (NPRA) manuals for road design [Veg08] describe road categories, road measures and design principles. For each specific road category there are various conditions that influence the aestetic design and layout of roads. A few example conditions are transport function, traffic load, safety and environment. Most prominent and visual differences in road design may be features such as road width, number of lanes, turns and lenghts. The former features are the most apparent design features, concerning the basic 3D structure.

In the scope of this project, the basic differences between roads,street and motorways are the most important. This is to keep the complexity and detail level down, but complexity should be a goal in mind for later potential extensions.

The following theory is adapted from [Veg08].

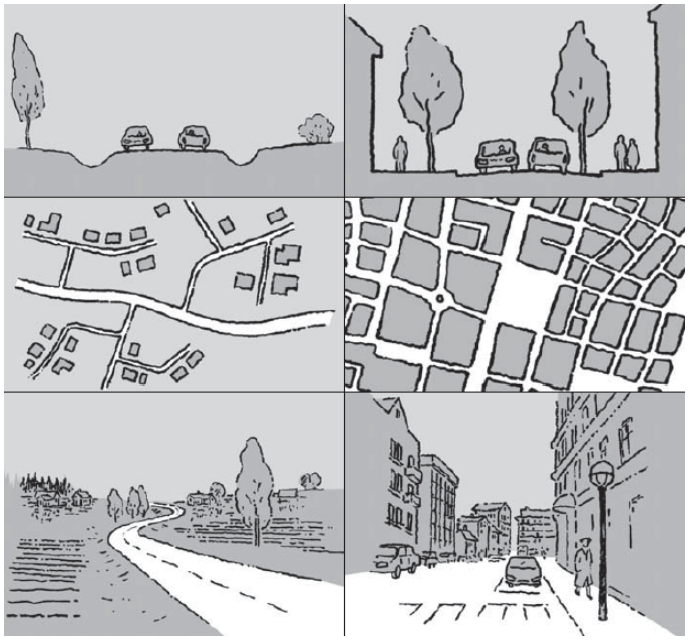

Figure 2.1: Common road and street differences. Road and street examples are shown in different settings in the left and right columns respectively [Veg08]. The first row show cross-section differences. The second row show macroscopic pattern differences. The third row show typical adjacent objects.

2.1.2 Roads

Roads are commonly found outside urban areas. Buildings are beside such roads are not rarely so close that they obstruct vision. See Figure 2.1. Roads' main purpose is moving heavy traffic, and often allow higher speed limits than streets.

Roads normally allows a speed limit between 50-100 km/h and maximum rise of 8% elevation. Roads may have several subcategories that each define subrestrictions for junctions and their exiting/entering roads.

For both streets and roads the cross section design is greatly affected by speed limit and traffic density. See Figure 2.2.

2.1.3 Streets

Streets are commonly found in urban areas, and may often contain mixed traffic, such as pedestrians and cyclists in addition to vehicles. This forces

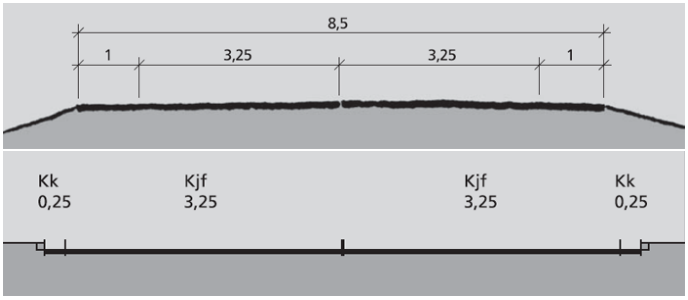

Figure 2.2: Shown are typical road and street cross sections. The top is a road cross section. The bottom is a street cross section. Units displayed are in meters. Adapted from [Veg08].

speed limits down, and often demand pavements that are adjacent to the streets. Streets may also be found in outer-city residental areas. Typically streets are developed after the principle of mixed traffic and crosses at grade, i.e. roads cross at the same level, with the need of for example, traffic lights.

Street normally demands a speed limit between 30-50 km/h and a maximum rise of 8% elevation. In practice these demands are difficult to satisfy, especially the elevation maximum as Norwegian terrains are rough to come by, so exceptions exsist.

2.1.4 Motorways

Motorways are roads of high speed limits whose main purpose is to lead traffic efficiently between major cities. Motorways have additional designs to mend high speed risks, such as seperate on- and off-ramps. The directional lanes are often seperated with a delimiter or space for safety reasons. Motorway junctions (interchanges) are typically designed to permit traffic on at least one motorway to pass through the junction without directly crossing any other traffic stream. Interchanges are often used when at least one of the roads is a limited-access divided motorway, but may also be used at junctions between two leveled streets. Interchanges are large and complex structures, but contribute to traffic flow efficiency and safety.

2.1.5 Junctions

A road junction is where two or more roads either meet or cross at grade (same level). Street crosses are often prone to accidents, and countermeasures are added such as physical narrowings and additional markings. T-junctions and X-junctions are the most common in streets, and often have pedestrian

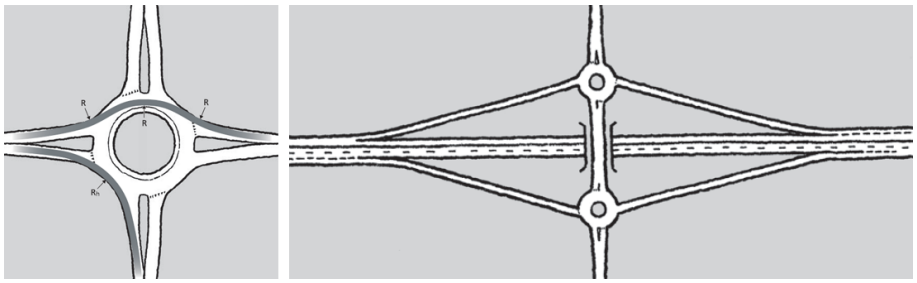

Figure 2.3: A common four-way roundabout on the left, with gray lines indicating two possible driving paths through it. On the right is a motorway overpass with roundabout junctions [Veg08].

crossings. Road junctions often have yield and channelised lanes in addition, as opposed to streets that have lower speeds. Roundabouts may replace either where applicable, and are often preferable with positive effects on safety, traffic flow and maintenance cost. Roundabouts differ from the other junctions, as they usually has a directed one-way cycle between the entering roads. Roundabouts are preferred when the traffic flow from all entering and exiting lanes are even. Note that T-junctions can also be called Y-junction or fork when it splits and resembles the Y shape. The overall design of junctions are greatly affected by the number of entering and exiting lanes.

2.1.6 Traffic Flow

Traffic flow is the flow of vehicles through a network. Three variables are required to visualise a traffic stream, which are speed, density and flow [GH75].

$$Flow = speed * density$$

The speed variable is the speed of vehicles or objects passing a point, or the average speed over a given length. The density variable is the number of vehicles passing a single cross-section of a network.

Network flow throughout a network can be measured by the maximum flow theorem, from the field of optimisation theory. Ford-Fulkerson is the first known maximum network flow algorithm, described in [For56]. This theorem measures the networks ability to move traffic from a source node to a sink node, over edges of the network. The requires edges to have an associated maximum capacity.

The maximum flow algorithm:

1) Search for a path from source to sink with still available capacity.

2A) If available capacity: Add to total flow the minimum remaining capacity, and subtract capacity from all edges in the path. Return to step 1.

2B)If no more available capacity, then return the total flow value.

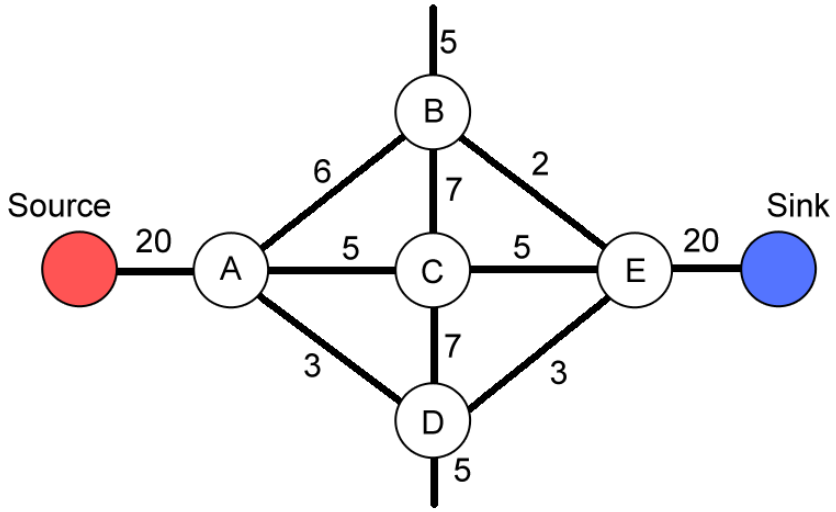

Figure 2.4: A maximum flow example. The flow goes from the source to the sink.

In Figure 2.4 a maximum flow problem example can be seen. The displayed circles represent nodes and the lines are undirected edges. The source gives out an infinite flow. The flow is limited by the maximum capacity, specified on every edge. Similar to road networks, this graph is a undirected graph, which means there is possible to flow either way over an edge. In this example, the maximum flow into the sink becomes 10, as the bottleneck is the edges BE, CE and DE.

This theorem can be extended into a multi-source and multi-sink problem by adding a consolidated source and sink, with imaginary edges of infinite or high capacity connected to all sink and source vertices respectively.

Figure 2.5 show an multi-sink example. In the example, the maximum flow becomes 19, where all the outgoing C (source) edges are the bottleneck.

It is also possible to put the capacity limit on the nodes of the network, where as outgoing thoughout can not be higher than the node capacity.

Note that there are more time and memory efficient variations available than Ford-Fulkerson, but are on the other hand much more complex in implementation. See [GR98] for a comprehensive list on complexity variations.

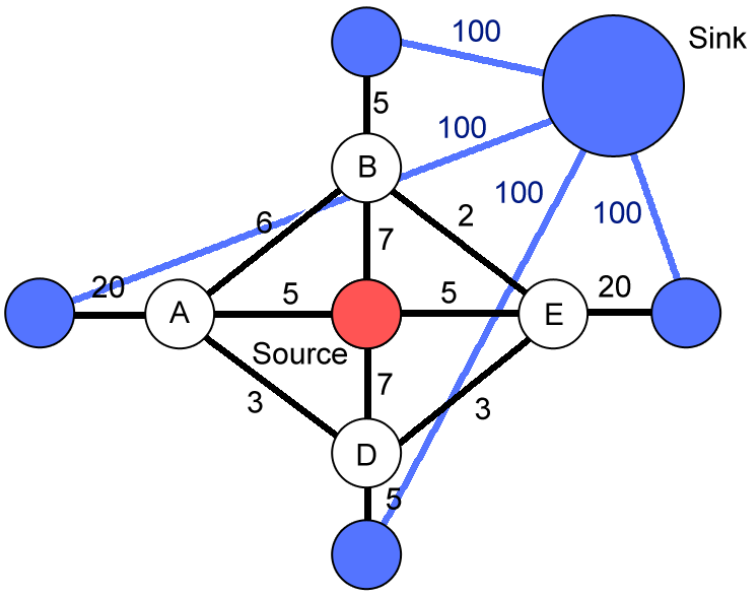

Figure 2.5: A maximum flow example. The flow goes from the source to the many sinks connected to the supersink.

2.2 L-systems

L-systems were first introduced in 1968 by Aristid Lindenmayer [Lin68], as a class of formal models called rewriting systems, to describe the developmental process. The rewriting systems were later adopted as 'L-systems' and is now a common developmental representation. The main benefit of such developmental representations provide means of defining potentially complex structures with compact descriptions. Other advantages may be scalability, self-organisation, robustness, adaptability and evolvability [FM08]. L-systems have been proven to excel on representing and simulating growth in 3D structures, especially plant ecosystems and organisms [PL90, PJM94, PHHM96] [DHL+98].

Since its introduction, L-systems have been extended with many variations. The following sections cover different L-systems used in this project. Most of them are adapted from the comprehensive book

The Algorithmic Beauty of Plants by the inventor [PL90].

An L-system is a rewriting system that operates on strings of *symbols*. The system is defined by assigning an *alphabet* of symbols, an initial string of symbols, and a set of rewriting rules. The initial string of symbols is also referred to as the *axiom*, whereas the rewriting rules are also called *productions*.

The productions specify how a symbol is replaced by a single or a string of symbols at each rewriting step. As symbols stem from a finite alphabet, a string of symbols is commonly refered to as a *word* [PL90]. The alphabet can be defined explicitly or implicitly in the set of productions.

One single iteration of the L-system consists of an attempt to rewrite every symbol with a production rule once.

The axiom represents the initial state of the L-system, and productions define the development of the string. The development is simulated by a sequence of discrete derivation steps. For each step, every symbol is rewritten in a conceptually parallel manner, using the first applicable production in the set. If no productions apply, the symbol rewrites into itself, simulating no change with what is called an *indentity production*.

The stopping condition on development can be a predefined number of rewriting steps or iterations, or the observation that the structure has not changed between two iterations.

Definition. An L-system in its simplest form is context-free and deterministic [PL90]. A deterministic 0L-system (0 notes zero-context) is defined as the ordered triple G = $\{\Sigma, \omega, P\}$ where:

$\Sigma = \{s1, ..., sn\}$ is the *alphabet* composed of a set of distinct symbols, s_i.

$\omega \in \Sigma$, is the non-empty *axiom*, made from symbols in Σ.

$P \subset \Sigma \times \Sigma*$, an endomorphism defined on $\Sigma*$, known as the finite set of *productions*.

A production (s, χ) $\in$ P is written in the form s $\rightarrow \chi$, where the symbol s is known as the *predecessor* and the word $\chi \in \Sigma*$ the *successor* of the production. An 0L-system is deterministic if, and only if, each symbols A in the alphabet, have exactly one corresponding production which gives $A \rightarrow \chi$.

An example of a deterministic 0L-system (D0L) with three iterations of G = $\{\Sigma, \omega, P\}$:

$G = \{\{A, B\}, AB, \{A \rightarrow AB, B \rightarrow A\}\}$:

ω: AB

i1: ABA

i2: ABAAB

i3: ABAABABA

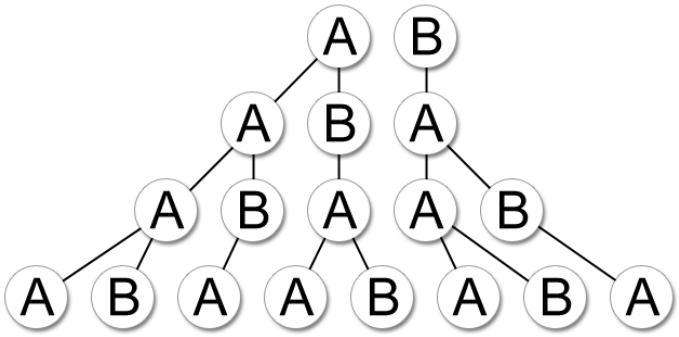

Figure 2.6: A derivation tree showing the development of an example axiom AB.

2.2.1 Bracketed

In bracketed L-systems, branching constants are used to start and stop branches, commonly noted with '[' and ']'. The '[' notes the start of a branch and the ']' notes the end. The start of a new branch indicate the start of a subgroup of symbols, and branches may exsist inside other branches. In topological representations, the branching symbols function as push/pop stack operators where the new branch will inherit the current position of its originating branch. It is possible for a branch to additionally inherit other attributes from its parent and / or adjacent symbols.

The following example shows a bracketed L-system in three iterations:

ω : A

P : $A \rightarrow B[A]$

$\mathrm{i}_{0:3}$: $A \Rightarrow B[A] \Rightarrow B[B[A]] \Rightarrow B[B[B[A]]]$

Other brackets such as curly brackets {} can be used to call external functions. Curly brackets can alternatively be used to prevent rewrite of the contents between the curly brackets.

2.2.2 Stochastic (s0L)

A stochastic L-system associate multiple production rules to a single predecessor symbol, where each production rule has an associated probability of firing. The set of associated production rules usually share a probabilistic sum of 1 for the same predecessor symbol. If multiple predecessor symbols exists in the production set, each symbol may have its own associated probabilistic sum of 1. In cases where there is a single production and a probability

bound to it, the rest sum of the probability is assumed to be the probability of an identity production.

A stochastic 0L-system is an ordered quadruplet $G_\pi = \{\Sigma, \omega, P, \pi\}$ with the extension $\pi : P \rightarrow (0, 1]$. π is the probabilistic distribution that maps a set of probabilities to a set of productions. In the example below the production probabilities are added at the end of the derivation symbol $\rightarrow$:

$\Sigma = A, [,]$

$\omega : A$

$p1 : A \rightarrow^{.5} A[A]$

$p2 : A \rightarrow^{.5} AA$

Two full iterations with the above rule set may result in one of the strings: $AAAA$, $A[A]AA$, $AAA[A]$, $A[A]A[A]$, $AA[AA]$, $A[A][A[A]]$, $A[A][AA]$ or $AA[A[A]]$

2.2.3 Parametric (p0L)

A parametric L-system extends symbols with the possiblity of holding multiple parameters in parantheses following a symbol. Formally, a parametric p0L-system is defined as an ordered quadruplet $G = \{\Sigma, \Pi, \omega, P\}$, extended with the set of formal parameters Π. Example parametric production rules:

$A(x) \rightarrow A(x + 1)B(x - 1)$

$A(x, y) \rightarrow B(x)C(y)$

where x, y belongs to a formal set of parameters Π.

Parametric symbols allow for further control of growth rates, which can be specified by parameters. For instance if the growth length is defined as x, a rule of type A(x/2) would halve the length for each derivation.

An additional degree of flexibility can be added through admitting conditions within the production rules, as in:

$$A(x) : x > 1 \rightarrow A(x + 1)B(x - 1)$$

where the production rule is applied only if the condition stated between the symbols : and $\rightarrow$ is true for the actual value of the given parameter x. It is possible to mix parametric and nonparametric rules. The symbol = may

be added to separate the predecessor, condition and successor. The compare symbols $< >$ must be set before the $=$ sign.

The conditionals may contain arithmetic operators $(+, -, *, /)$, logical operators $(\&, !, |)$, relational operators $(>, <, >=, <=, =)$ and parentheses ( ), which all can be added to the set of formal parameters Π.

A production matches a module in a parametric word if the following conditions are met [PL90]:

1. the symbol in the module and the symbol in the production predecessor are the same
2. the number of actual parameters in the module is equal to the number of formal parameters in the production predecessor
3. the condition is true if the actual parameter values are substituted for the formal parameters in the production.

2.2.4 Context sensitive (IL)

Context-free L-systems ignore the context in which a symbol appears. Context sensitivity can be added to the production rules with the symbol $<$ for left context and $>$ for right context. Context sensitive L-systems can be divided into two classes; 1L-system and 2L-system. The 1L-systems considers either the left or the right context of a symbol. The 2L-systems considers both left and right context.

$$left\ context < strict\ predecessor > right\ context \rightarrow successor$$

1L-systems have one-sided context only and are in either of the forms:

$$B_{Left} < A \rightarrow \chi \text{ or } A > B_{Right} \rightarrow \chi$$

The following is a signal propagation example to show how context sensitivity can simulate signals, adapted from [PL90]:

$\omega : BAAAA$

$p1 : B < A \rightarrow B$

$p2 : B \rightarrow A$

The resulting word in two iterations:

$$BAAAA \Rightarrow ABAAA \Rightarrow AABAA$$

Note the signal b in the example above, that moves across the string through iterations.

When combining context-sensitivity with brackets, the brackets originating from the same level as the strict predecessor can be ignored. In the example below the production $p1$ applies to the symbol a in the axiom:

p1 : $B < A > C[D]E \rightarrow G$

ω : $UB[V[WX]]AC[DY]EZ$

Additionally, context-sensitivity can be combined with parametric L-systems. Example parametric context-sensitive production rule:

$$A(x) < B(y) > C(z) \rightarrow F(x, y, z)$$

2.3 Interpretation of L-systems

The symbol topology of L-systems is semantically agnostic, which means that without interpretation an L-system remains an abstraction of structural transformation. Interpretations differ greatly depending on the field and need of output. An interpretation can generate visual graphics [PL90] or even music [Man06].

For each application a parser is needed to map the string output in a meaningful way. For describing geometric structures, such as 2D/3D computer graphics, the following pipeline may be used:

L-system $\Rightarrow$ word $\Rightarrow$ graphic interpreter $\Rightarrow$ geometric structure

The most basic and common graphical interpretation of L-system strings, called *turtle interpretation*, is described by [Pru86]. The principle of turtle interpretation is imagining a turtle seen from above, walking about and drawing a line in the path it has covered. The turtle has a position in Euclidean space, represented by the Cartesian coordinates x,y, and an orientation angle α. Given the initial parameters (x_0, y_0, α_0), with start coordinates x_0, y_0 , the initial heading α_0, a stepsize d and an angle increment δ , the turtle can be moved with the following symbols and constants:

F : move forward with the length of d. The turtle moves to the point (x', y', α), where $x' = x + d\ cos(\alpha)$ and $y' = y + d\ sin(\alpha)$. The result is a drawn line segment between points (x,y) and (x',y').

f : same as above, but without the drawing of the segment.

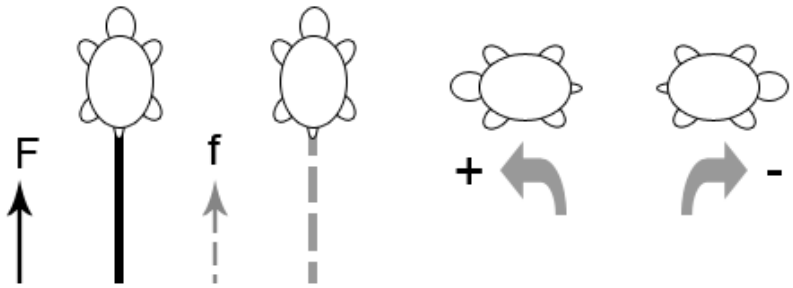

Figure 2.7: Turtle interpretation movement.

+ : the turtle turns left, with the angle δ. The turtle is now facing $(x, y, \alpha + \delta)$.

- : the turtle turns right, with the angle δ. The turtle is now facing $(x, y, \alpha - \delta)$.

For example, the string F+F+F+F would display a closed square when interpreted with increment $\delta = \frac{\pi}{2}$ (90 degrees).

The interpreter interprets symbols one at a time as they are found in the symbol string. Symbols not in the interpreter set will not be interpreted, which permits use of auxiliary symbols that help define the growth process without interfering with the graphical interpretation [FM08].

The turtle interpreter also interprets branching, with the corresponding symbol meanings:

[- Saves the current state of the turtle onto a pushdown stack.

] - Restores a state from the stack and makes it the current state of the turtle (last-in first-out). No line is drawn, but the position and angle of the turtle changes.

In Figure 2.8, various plant geometry, drawn by using bracketed turtle interpretation, can be seen. Under each plant is the number of iterations and turning angles specified, along with the used axiom and productions.

The basic turtle interpretation can also be extended to form 3D models [PL90].

When using parametric L-systems, the moving and turning of the turtle in interpretation can be set by modules:

$F(\alpha)$ - move forward with α distance/steps and draw a line.

$f(\alpha)$ - move forward with α distance/steps and do not draw a line.

Figure 2.8: A turtle interpretation visualising plant geometry, adapted from [PL90]

$R(\alpha)$ - rotate the turtle clockwise with angle α.

$L(\alpha)$ - rotate the turtle counter-clockwise with angle α.

This approach avoids the problem of self-cancelling and unneccesary long strings, such as: $F++++---+---+---+F$

2.4 Genetic Algorithm

Genetic algorithms (GA) were introduced by [Hol75] and are a sub-class of evolutionary algorithms (EA). Common to the class of EAs is the use of techniques inspired by natural evolution to generate and find solutions to optimisation problems.

2.4.1 Overview

The canonical genetic algorithm consists of a population of strings (genotypes), which encode candidate solutions (phenotypes) to an optimisation problem. The candidates are combined and evolved in generations. In each generation, the *fitness* of every string *individual* in the population is evaluated. Multiple individuals are stochastically selected from the current

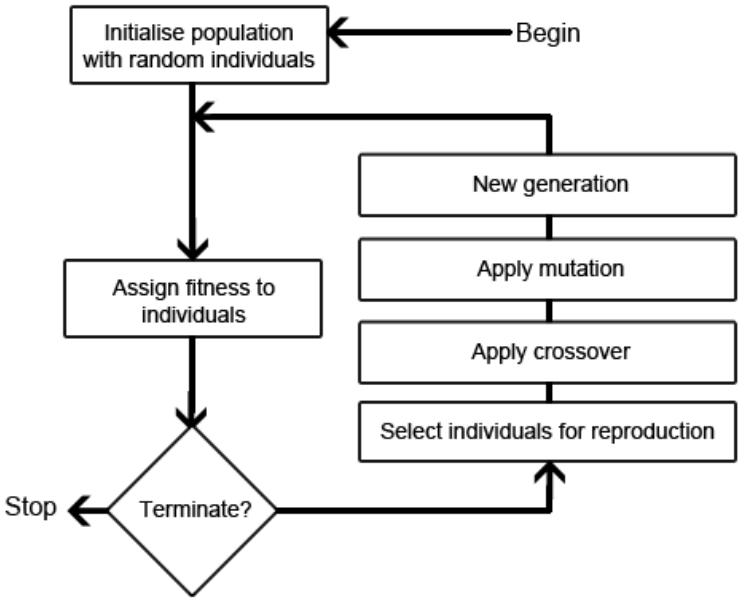

Figure 2.9: A simple genetic algorithm flowchart.

population, with a higher chance of being selected with higher fitness. The selected individuals have their gene strings recombined and/or mutated to form a new population. The new population, as a result of a generation, is used as input for the next iteration round of the algorithm. The algorithm loops until it terminates due to a stopping critera. See Figure 2.9 for the common GA loop. The main steps of a GA are described in the following sections.

2.4.2 Initialisation

Traditionally individuals are randomly generated to form the initial population. That means the strings are built by randomly combining allowed values. The size of the population depends on the problem domain, and could be a single individual to tens, hundreds or thousands of individuals. Alternatively the solutions can be seeded or aided into search space areas where the optimal solution is more likely to be found.

Candidates are encoded as genetic representations, often in binary code as strings of 0 and 1, but other encodings such as character, number or tree-based are possible. Below is a binary code encoding example, with corresponding decoded number phenotype:

Genotype example : 1100 0110 1010

Phenotype example : 12 , 6 , 10

2.4.3 Selection

During a single generation's lifetime a group of individuals will be selected for reproduction. The selection is a fitness-based process, where the fittest inidividuals have the higher chance of being chosen. There are many variations of fitness-functions used to perform the selection. The generic fitness function credits a fitness value to each individual solution, which afterwards is normalised over the whole population.

The following list describe a few common selection mechanisms adopted from [FM08]:

Fitness proportionate selection , also known as roulette-wheel selection, assigns a probability to each individual which corresponds to the ratio between the individuals fitness value and the sum of all fitness values in the population.

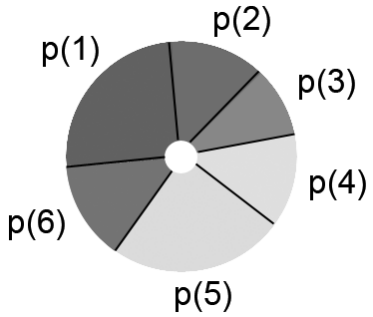

Figure 2.10: Fitness proportionate roulette-wheel example. In this example the probability p is highest for individual 5.

$$p(i) = \frac{f(i)}{\sum_i^N f(i)}$$

The probability $p(i)$ that an individual i is selected, is based on the ratio between the individual fitness $f(i)$ and the sum of the fitness in the population (see Figure 2.10). Offspring is selected by spinning the roulette wheel N times, where N is the population size or the number of offspring to be produced.

Rank-based selection assigns a rank to every individual based on its fitness value. The higher rank will always have the slightly higher probability of being selected than the succeeding rank. This selection evens out the fitness differences, and helps hinder the search to narrow down too quickly, as might happen with fitness proportionate selection.

Rank $10 > 9 > 8 > 7 > 6 > 5 > 4 > 3 > 2 > 1$

Tournament selection involves running "tournaments" with sub-sets of k individuals chosen at random from the population. k is known as the tournament size, and the selection pressure is adjusted by changing the tournament size. The winner individual of each tournament (the one with the best fitness) is selected as a parent. This continues until the sufficient number of parents is selected. Tournament selection achieves a good compromise in maintaining both selection pressure and genetic diversity in the population [FM08].

Elitism can additionally be used to automatically push forward the best individual of the population into the next generation directly. Elitism works as a safety mechanism to ensure that the single best solution is not lost from generation to generation.

2.4.4 Reproduction

After a group of individuals are selected for breeding they reproduce new population members using genetic operators. Common genetic operators are crossover and mutation. Parents are chosen in pairs to produce children/offsprings. The offspring will inherit any characteristics from its parents determined by a crossover function. Mutation will result in a new gene which is not directly inherited. The reproduction may continue until an entire new population is filled, or until all parents have bred. The new population can either be merged with the existing population or entirely replace it.

Crossover Crossover functions crosses the gene strings of the parents into a new string for the offspring. The crossover might be one-point, multi-point or uniform crossover. The offspring should at least have genes from each parent, otherwise it will be a copy of either. One-point crossover simply divides both parent's genotype in two parts and exchange one part with the other to create two new genotypes, each with a part from each parent. Multi-point crossover divides the parents genotype into n segments and exchange corresponding segments. Uniform crossover exchange the genetic content at n random chosen positions. This crossover can be used on modules, exchanging

modules one-to-one. A single point crossover example can be seen in Figure 2.11 where two parents do a crossover and create two offspring whom are a mix of both parents.

Figure 2.11: Shown above is a single point crossover. The two parents on the left cross over and produce the offspring on the right both having a mix of parent genes, split by the point.

Mutation Mutation is the random alteration of a gene during the production of offsprings. The mutation might be limited to a chance of only one gene in total being altered, which would be a minimal mutation. The mutation operator must be adapted to the genetic representation used, such as binary, real number, symbol or tree-based. The mutation operator must be designed to reach every possible value in the genetic representation space. Mutation helps to avoid local minima, but too high mutation may scramble the search and work against its purpose.

Example string before mutation:
1100 1011 0110 1001
Example string after mutation:
1100 1**1**11 0110 1001

2.4.5 Termination

Common termination criterias are:

- a fixed number of generations
- a found optimal solution
- an observation that the highest ranking fitness has hit a plateau and stagnated (local minima).

If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached.

2.5 Motivation

The wide motivation behind this thesis is the need of automatic procedural generation, often found in computer graphic fields, where very high detailed models come closer and closer to reality. Constructing large,complex and highly detailed models is very time-consuming by hand, and the benefit of automatisation is quite clear. The Norwegian Public Roads Administration expressed the same need of automisation of content for their simulation environment and studio application. This situation motivated for research towards finding an automation alternative fit for their purpose and on Norwegian road network models.

Parish and Müller's article on Procedural Modeling of Cities [PM01] is one of few publications covering automatic road generation using L-systems. Their work is the main motivator for using L-systems to generate road networks in this thesis. The work of [PM01] is, in summation, the most relevant work to this thesis, and close to the problem at hand.

The proposed system in [PM01] uses a procedural approach based on L-systems to model cities, and thereunder connected road networks. For their system to create cities, various input image maps are given as inputs, such as land-water boundaries and population density. Based on these inputs the system generates a system of roads and streets, divides the land areas into lots, and creates the appropriate geometry for urban buildings on the respective allotments. For the generation of transport networks, their L-system is extended with a module that allows the consideration of global goals and local constraints such as external functions. Locally these functions will prune, extend or merge road segments into nearby segments, to make crosses/junctions and road patterns. An impressive reinvention of Manhattan done by their system can be seen in Figure 2.12.

Parish and Müller's system succeed in generating good basic road networks, but the realism in the model is still far from simulation quality. Especially the junctions are far from realistic, and there are no roundabouts. Additionally their approach need global parameters, patterns and layers of images as input. As with [PM01] and other L-system applications, there is much tuning of initial parameters to be done to get the correct output.

The multiple inputs needed for Parish and Müller's system, motivated a search for other possiblities to shape road networks. Genetic algorithms are well known means of optimisation, and have the bonus of being bio-inspired as with L-systems. This motivated for a evolutionary L-system solution.

Several works show combining genetic algorithms with L-systems can be successful, when tuning parameters by scoring solutions during run-time [NSW03] [RLFS02] [RCB+02] [HP01] [Och98].

Figure 2.12: An example road network generated by an L-system. The images are adapted from [PM01]. The top image shows a road network generated inside the Manhattan island contour with specified patterns. The bottom image shows a drawn version of the actual Manhattan road network.

[Och98] show that it is possible to count and use aesthetic features as fitness, which speaks in favour of road networks where you have statutory and de-facto rules that define the structure. Ochoa applies multiple fitness measures on plants drawn by L-systems, in an attempt to eliminate human participation when picking phenotypic traits. This motivates for applying similar functions to road networks. Also in [RLFS02] the authors were faced with the difficulty of constructing suitable L-system production rules for generating a leaf shape. By specifying the contour of the leaf, they successfully drew a leaf form by combining GA and L-system. See Figure 2.13

The authors of [PM01] state that since there is no underlying model of transportation flow simulation on their roadmap, they can ignore the im-

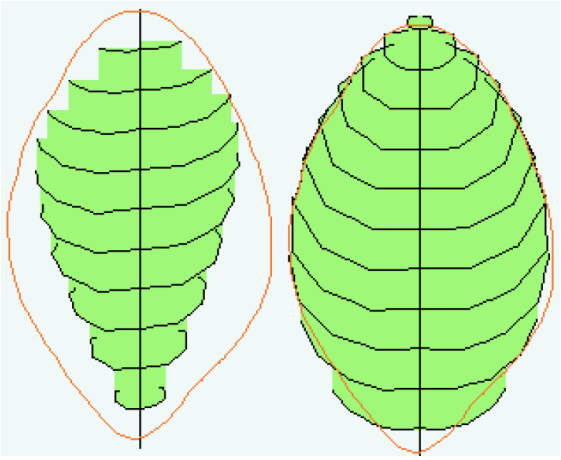

Figure 2.13: An example leaf generated by an L-system tuned by a genetic algorithm. The images are adapted from [RLFS02]. The left hand image show the first generation of an GA tuning an L-system that draws a leaf shape.. The right hand image show the best optimised leaf after 200 generations. Images are adapted from [RLFS02].

plications on street capacity. In road networks used for simulation, such implications must be addressed or eventually be used as a measure of quality. This motivated for experiments with flow measures, to optimise a road network into a higher detail level than just by L-system geometry.

Chapter 3

Architecture / Model

The model is implemented as a framework with a high degree of flexibility. The framework consists of several modules that communicate with each other. Each main concept is implemented as an individual module. This means there is one module for each of the L-system, the genetic algorithm, the interpreter and the flow algorithm. The following sections elaborate on the various modules.

3.1 L-system

The L-system used to represent a road network is kept simple, and is designed to take advantage of the simplicity of the common turtle interpretation. The main alphabet of this L-system is, but not limited to:

F,+,-,#,[,],(,)

The above alphabet consists of both variables and constants, and some symbols that can be both, depending on the context it appears. The underscore replaces the minus sign in the alphabet, as to avoid confusion with negative values in parameters. Positive values do not have a prefix '+', i.e. values without a prefix are positive. The variables of the alphabet are treated as parametric modules, and may contain numeric or constant parameters. Variables are implemented as symbol modules, which means that each symbol is an unique object and may hold several hidden parameters. This allows for flexibility towards keeping pointers to neighbouring symbol modules. Additionally does symbolic and synthatic encoding ease use and implementation [Och98] [Jac01] [RCB+02].

The F symbol is pre-set in the interpreter and represents a road segment, which corresponds to a line as in the common turtle interpretation. All symbol modules may also have multiple parameters in succeeding parantheses,

such as: F(1,2,3). These parameters may define and take advantage of both interpretation features and production rule features.

Branching of road segments are described by brackets [and]. Any symbols that appear between brackets belong to a new branch. The way this is implemented is that the first bracket [is implemented as a module, which may have children (a sub-string of modules), and the] is a constant that notes the end of the sub-string. This additionally allows for the branch to have parameters, which are listed in parantheses behind the end constant as such:

$$F[F](1,2,3)F$$

Though branches are independent in the string, the geometric location for the stem origin of the branch is set by the preceeding module or symbol, i.e. $F[\ldots]$ the branch stems from the end position of F.

The L-system must be initiated with one to multiple rules, an iteration number and an axiom. Symbols not specified in the alphabet, besides [,],+,−, will not be added to the interpreters search alphabet.

In this framework a mix between stochastic, parametric and deterministic rules may be used. Up to 2L-context sensitivity may be used, that is; 0L,1L or 2L context-sensitivity.

When using multiple rules that share a precedessor, they are grouped to share a probability of 1 (100%). In example, three production rules with a shared predecessor F will be assigned a 33,333...% probability and be grouped, forcing atleast one to be applied. If a stochastic production rule does not share a predecessor symbol with any other rule, the remaining probability of 1 counts for the probability of the identity production being applied.

Once a production rule fires for a symbol, the symbol is replaced and the iteration continues to the next symbol.

A special twist to the implementation is the use of empty parentheses and empty brackets. They note that the successor with the empty clauses should copy and aquire all of the precedessors parameters or children.

Empty parentheses example:

Production rule: $A \rightarrow B()$

In use: $A(1,2,3) \rightarrow B(1,2,3)$

Empty brackets example:

Production rule: $A \rightarrow D[]$

In use: $A[BC] \rightarrow D[BC]$

Stochastic production rules would in a true random implementation produce different results on each run. To make runs reproducible a random seed can be set. This pseudo-random seed is implemented in the production rule picking process, which will make stochastic productions fire at the same frequency on consecutive runs. The default seed value is 0.

3.2 Interpreter

The interpreter interprets the L-system string into nodes and edges with Cartesian coordinates (2D). The road segments, the F symbols in the L-system, are drawn in their simplest form, as a line. From the L-system F symbol, the interpreter creates an segment object, which hold values beyond what is visible in the simplified 2D interpretation. The line is the only visible part of a road segments in interpretation, but the road segment object may still hold multiple hidden parameters, describing road length, neighbours, number of lanes, and more. The curvning of segments is ignored in this implementation, as two cartesian coordinates are only a function away from being a polynomial curve. Figure 3.1 shows two ways of drawing a curved road segment on a line with two endpoints.

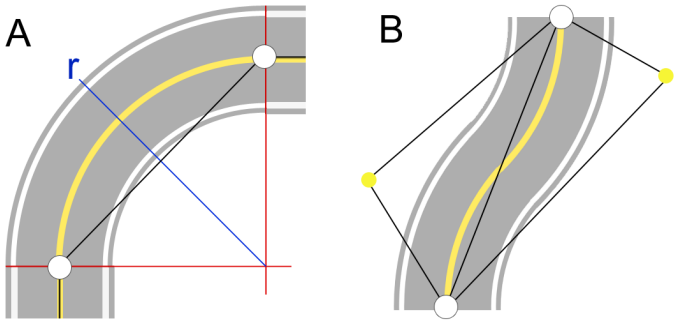

Figure 3.1: Shown above is a partial circle curve and a polynomial curve.

The interpreter is initiated with a vector containing a starting point and a starting direction. The direction is in the interval $[0, 360]$, and is initially set to 0 (east).

In example: a F symbol equals a straight line, whereas if a parameter is specified such as $F(90)$, the current direction is changed by rotating the line around its starting point with 90 degrees. A negative parameter value would rotate clockwise, and a positive value would rotate counter-clockwise.

A constant parameter such as # is used to rotate a random amount of degrees upon interpretation. This random value's range can be preset in the interpretation, in example $[30, 150]$ to prevent too small angles on branching segments and segment chains curling in on themselves. As with the stochastic L-systems, this random turning behavior can be subject of a pseudo-random seed, instead of true randomness, to be able to produce the same result on consecutive runs on the same L-system string. The default seed value is 0.

Once the interpreter has interpreted the entire L-system string, it continues to search for intersections. To find and generate intersections as objects, the interpreter searches the list of segments after line-line intersections. The interpreter splits found intersecting segment pairs at the intersection point, creating up to four new segments. Once no segments intersect each other beside at the end points, the interpreter creates intersection objects where three or more segment endpoints meet. Each intersection object holds pointers to connected segments. Each segment has a connection to up to two other segment endpoints.

For the flow network phase, additional information needs to be extracted from the network. For this phase the interpreter prepares the intersections by following neighbouring segments from each intersection. This maps connected intersections to each other, and adds an imaginary edge between them, which represents the connected path between them.

3.3 Genetic Algorithm

A genetic algorithm (GA) is used to find a set of parameter values that will be used as input to the candidate L-systems. The same GA is also used in optimising intersections for maximum flow. The main difference is in the fitness functions used.

Individuals of the GA carry the genome parameters, a simple list of integers that are in a given range. For the first phase, the range is normalised to $[0, 100]$ which correlates to a precision of 0,01 or 1% percent if the parameter is used as a probability value. For the second phase, the range is set between the number of intersection types that is specified.

Each individual of the GA population consists of an L-system, or a link to an L-system template, and a parameter chain. The L-system of each individual have the same features and a common interpreter. The L-system is a template of production rules, axiom and alphabet, to give each individual the same building blocks.

Fitness For the L-system tuning phase,the fitness score of an individual is determined by applying its genome parameters to the L-system, and measuring features in the interpretation domain. For the flow network tuning, there is only one fitness function that is applying intersections to the interpreted network and measuring maximum flow.

The following are possible fitness features that can be activated in the framework for the L-system tuning:

- Road length. The sum of road segment lengths combined can be used as a fitness feature. This favoures L-systems that create road networks with higher total length. If the parameters of the genome directly influence the length of each segment as a symbol parameter F(length), this fitness has a great impact. This fitness value increases at a rate: number of segments * segment length.

- Intersections. This fitness function measures the number of intersecting road segments in the interpretation. The way this is measured is by comparing each segment with every other, to see if they intersect between their endpoints given two sets of x,y coordinates. When comparing, a segment can only intersect an other segment once, at any one point. When an intersection is found, the two segments are divided into four new segments. Segments that meet in their endpoints get converted into interpreted intersections (junctions).

- Intersection-Segment Ratio. This function is essentially a mix of the two preceding functions. This ratio would reflect if the road network creates blocks, large or small, or spreads without generating intersections. In example, 1 minimal single block would have 4 segments and 0 intersections. 4 adjacent blocks could have 10 segments and 5 intersections. In result, an urban neighbourhood would have a intersection-segment ratio of approximately 1:2.

- Closed loops. The measure of closed loops can be used to determine the realism of the road network. Loops, or lots, are common in road networks. In cities, blocks and loops are common, encapsulated by 3 or more intersections. Lots closed in by long roads on the country side would be much larger. The number of fully closed loops, and their average length case be used to reflect wether the created network as an accessability and block patterns.

- Network circumference. The total approximate circumference of the city can be used as measure of how widespread the network is. The total

distance around the interpreted network's outer points is calculated. This is easier to calculate when intersections have been found and the difference between inner and outer points are clear. This fitness would prefer a road network that is evenly spread out, and not in a narrow long shape.

- High degree intersections. Very high degree intersections may arise depending on the rule set used. High degree intersections are not very realistic and should recieve a penalty. Combining this measure with a negative weight gives a penalty for every found intersection above the degree of 4.

Any number of the fitness functions that are activated are combined into one formula. Weights w are added to allow setting the importance of each contributing feature. Below is the final fitness formula shown with f_n as a function, with corresponding weight w_n.

$$\text{Fitness} = \frac{f1*w1+f2*w2+f3*w3+f4*w4+f5*w5+f6*w6}{w1+w2+w3+w4+w5+w6}$$

As the different fitness functions give out very different value ranges, care must be given to the weights, but also concerning the importance of each fitness function used.

Selection The fitness is used in the chosen selection mechanism, which can be any one of the three specified in Chapter 2. For the tournament selection a tournament size has to be additionally specified, which defines the number of individuals pulled into a sub-tournament. Depending on how the fitness function values arise, different selection mechanisms may be used with different outcomes.

Crossover The selected parents are put through a crossover mechanism, combining genome parameters from two parents, which results in two children. With a small genome, 1 crossover point can be used, but larger genomes benefit from multiple or uniform crossover points. For small genomes the crossover mechanism can be skipped, as long as there is sufficient mutation, which would produce similar results that of crossover. The crossover is controlled by a set rate, which corresponds to how many percent of the population should go forth to recombination, in range 0-100

Mutation The mutation rate is a probability for each genome parameter to mutate. A mutation range must be set, in where the mutation can mutate within. The allowed value range of the genome parameters will dictate the resulting value after mutation, if the mutation happen to land outside the allowed range then the value will be cropped.

Termination A maximum number of generations can be specified to halt the loop, if a maximum fitness/sought fitness (perfect solution) individual is not found, which is most likely when dealing with road networks.

3.4 Maximum flow

The maximum flow module takes an interpreted L-system as input. From this the module finds all intersections that are linked to each other. The module adds multiple sink intersections on lone edges around the road network. These become the exiting roads of the network.

A single high-degree intersection is selected as the source intersection. The implementation consists of a simple model using only two different intersection types, where one is more effective than the other. These two types are randomly assigned on intersections in the network, and capacities are computed based on the speed limit of segments times the modifier specified by the intersection type the segments are leaving. The default settings for the types are: 1 = 0.5 multiplier and 2 = 0.75 multiplier. This means, for every intersection the the throughput traffic flow is reduced by the multiplier. This simulates crosses where hinders such as crossing traffic and traffic lights affect the flow.

This allows for a genetic algorithm to optimise the series of paths spreading from the picked source to the sink. These paths will effectively be the main roads leading in and out of the city. In this simple model, the road segments get initially assigned a default 30 speed limit, so that the capacity gets modified by the type of connecting intersection only. This leaves the density from Section 2.1.6 at 1.

Chapter 4

Experiments and Results

This section describes the experimental setup and plan, and continues to list experiments and results. The experiments are presented one by one and the results are successively discussed.

4.1 Experimental Plan

The experimental plan for this project is very straight forward, and can be broken down to milestones outlined by the research questions. The research questions are divided into three phases, as the presence of, and the genetic algorithms vary between these phases.

To sum up the research questions, divided in phases:

Phase 1

- What are the minimal and required features of an L-system to create valid road network models?

Phase 2

- Can an L-system be guided by a genetic algorithm in search of better automation of road network models.
- What fitness measures can guide the genetic algorithm?

Phase 3

- Will a maximum flow-driven GA be able to tune flow transitions in the interpreted road network?

4.2 Experimental Setup

The framework is implemented in Python and run on a MacBook from 2009 running with a 2,26 GHz dual-core processor and with 8 GB of memory. The input parameters to the system vary from experiment to experiment, and is described at each experiment where it differs.

This implementation uses Python version 2.7.1 to run the algorithms, and the Matplotlib library to visualise the results.

The shared input parameters to the experiments are:

Interpreter settings

Initial direction: 0 degrees (east)

Straight length (F): 10 units

Random angle (#): range $[30, 150]$ in CC or CCW direction

4.3 Experimental Results

The experimental results are presented in sections, divided by progression and included modules. The experiments do loosely follow the research questions in a chronological manner.

4.3.1 Experiments Part 1 - L-System

Early experiments showed that a simple deterministic L-system was decent for generating very basic road networks. This could already be seen in related work, such as [PL90], where plants already resemble road networks, but this was the first milestone. The simplest L-system consists of only three rules, with the actions branch/stretch/turn which are the main components needed to make a road network.

D0L-system :

Axiom : X

Rule 1 : $F \rightarrow F\#X$

Rule 2 : $X \rightarrow F[Y]Y$

Rule 3 : $Y \rightarrow \#F$

After 5 full iterations:

$$F\#X\#F[Y]Y\#F\#X[\#F]\#F\#F\#X\#F[Y]Y[\#F\#X]\#\#F\#X$$

$$[\#F\#X\#F[Y]Y\#F\#X[\#F]\#F]\#\#\#[\#F]\#F\#X\#F[Y]Y\#F\#X[\#F]\#F$$

The main drawback of this rule set is its deteministic nature and that a branching pattern will be repeated through the string, making intersections appear at a given frequency. Note that the derivation is deterministic, but the interpreted angle # is stochastic between the set value range. A drawback is the redundant constants # that appear in series, which replace each other in the interpretation and counts only to excess computation.

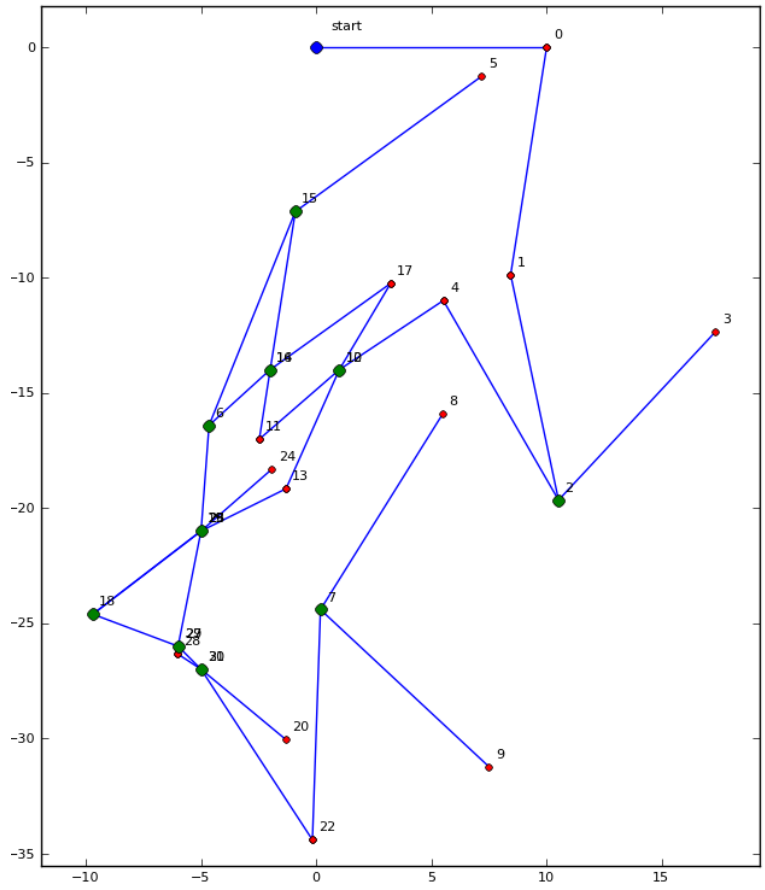

Figure 4.1: Figure shows a graph representing a road network. A simple deterministic L-system is used, where intersections are found by searching in the interpretation domain. Intersections are noted with green circles.

From Figure 4.1 one can see the resulting interpreted network resembles a map for a road network. Four-way intersections (or higher degree) appear when lines cross in the interpretation domain.

By adding the turning action # as a parameter of the road straight module F, the redundancy of the # constant is removed.

D0L-system 1 :

Axiom : X

Rule 1 : $F \rightarrow FX$

Rule 2 : $Y \rightarrow F(\#)$

Rule 3 : $X \rightarrow F[Y]Y$

Through 5 full iterations:

$F(\#)XF[Y]YF(\#)X[F(\#)]F(\#)F(\#)XF[Y]Y[F(\#)X]F(\#)\ \ldots$

The above setup give the same interpretation results as the previous set-up. Experiments with the deterministic rule set show a need for stochastic behavior to get a more natural look on the road network. Note the triple dots mark that the L-system word continues beyond.

SP0L-system 2 :

Axiom : F

Rule 1 : $F \rightarrow F(\#)F(\#)$

Rule 2 : $F \rightarrow F(\#)[F(\#)]$

After 5 full iterations:

$F(\#)[F(\#)][F(\#)F(\#)][F(\#)[F(\#)][F(\#)[F(\#)]]]\ \ldots$

The system above use a set of rules which share a stochastic probability of 0.5. The stochastic feature of the rule set make intersections appear of the structure $F[F][F]\ldots$. The drawback is that since the branch is rooted to the predecessor F module, more branches than 2 might appear, giving intersections of degree 4 and higher, which are rare in real road networks.

SP2L-system :

Axiom : F

Rule 1 : $F > () \rightarrow F()F(\#)$

Rule 2 : $F > F \rightarrow F()[F(\#)]$

Rule 3 : $F < [] > F \rightarrow 0.2[][F(\#)]$

After 5 full iterations:

$$F[F(\#)]F(\#)F(\#)F(\#)[F(\#)]F(\#)F(\#)[F(\#)F(\#)][F(\#)]F(\#)[F(\#)] \ldots$$

To mend the high degree branching and to keep a stochastic nature, the above changes was made to the rule set. The first rule applies only to end stubs, that is, F modules at the end of a substring inside a branch, or the F module at the very end of the entire string. The two first rules share a probability of 0.5. The second rule only applies to a F module, followed by a F module, in attempt to hinder the rule rewriting is own structure such as: "F[]" to "F[][]". The third and last rule is a rule designed to extend 3-way crosses into 4-way crosses. It also has an additional overriding probability to prevent it from firing too frequently. The uniqueness of implementing the bracket as a module, allows it to have children (and let them be moved entirely as a list), and double-sided context to both its endpoints which is a strong benefit.

The stochastic probabilities of this rule set can be used to create different road networks, concerning branching degrees, and road-segment-to-intersection ratio.

SP2L-system :

Axiom : F

Rule 1 : $F > () \rightarrow F()F(\#)$

Rule 2 : $F > F \rightarrow F()[F(\#)]$

Rule 3 : $F > F \rightarrow F()X$

Rule 4 : $F < X > () \rightarrow [F(90)X][F(-90)X]F()$

After experimenting with the previous setups, it proved difficult to make squared block-patterns emerge in clusters, as with real dense urban cities. By adding the rule $F < X >'' \rightarrow [F(90)X][F(-90)X]F()$ and a trigger constant X, the system is able to create urban block patterns.

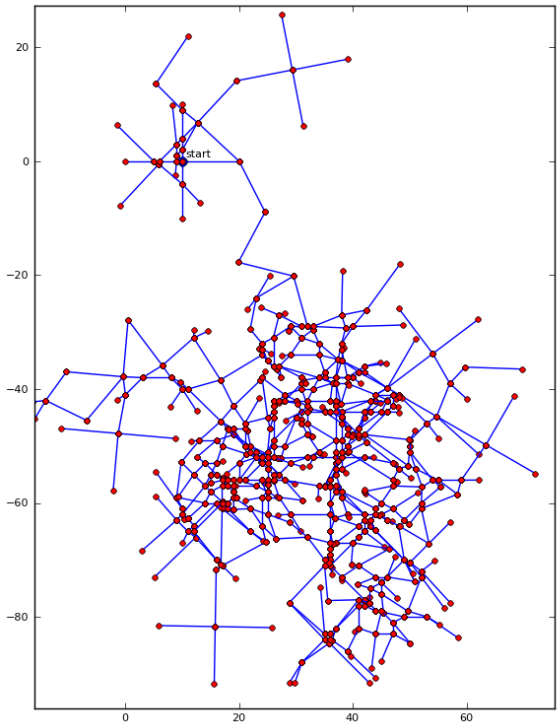

Figure 4.2: The network shown above has a city block sub-pattern generated by adding a single special rule, distinct from previous networks. Intersections are not marked by green circles in this figure.

Figure 4.2 shows the pattern made in the network. This rule set allows for city centers to emerge at different locations in the network. This is achieved by setting a small chance for the trigger symbol to appear, and then a high chance of replacing it with the pattern rule. In this way the pattern repeats itself outwards from the source. A drawback with this very rule is that it circles in on itself with the same turning angle and segment length, making redundant overwrites of itself in interpretation. These redundant segments are removed under interpretation as a special case.

The interpreted road network as it is displayed here, has no set width or scale on the road segments. This means that some spaces between connected road segments (inner polygons) might be too small compared to the overall set scale.

4.3.2 Experiments Part 2 - GA + L-System

Initial experiments showed that stochastic L-systems are hard to measure consistently, as the same set of rules can give quite diverse results. Using seeded random probability in the L-system and interpretation proved to give consistent results. 1 The following experiment was run with the following L-system and GA setup:

SP0L-system :

Axiom : F

Rule 1 : $F > () \rightarrow F()F(\#)$

Rule 1 : $F > F \rightarrow F()[F(\#)]$

Rule 1 : $F > F \rightarrow F()X$

Rule 1 : $F < X > () \rightarrow [F(90)X][F(-90)X]F()$

GA :

Population size : 50

Generations : 100

Genome length : 4

Crossover rate : 50 %

Mutation rate : 25 %

Selection type : Rank

For this setup a full mutation range was used. Only one fitness function was used, which measured the total length of the interpreted network.

Figure 4.3 shows the resulting fitness graph of the GA after doing a seeded run. A high fitness individual is early found and held for many generations until a better individual is found. The average fitness keeps steady throughout the generations. This behavoir might be from the fact that crossovers split the shared probability of the rules sharing a predecessor.

Seeded - Last generation :

Generation 100, Average fitness 106.80, Top fitness 940.00

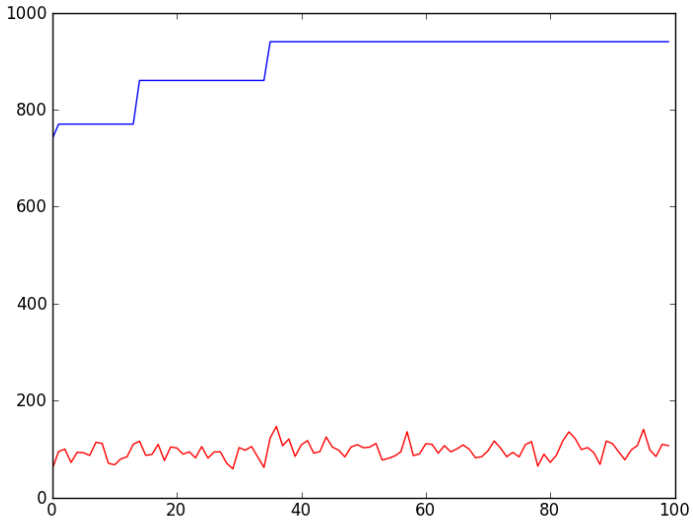

Figure 4.3: Fitness graph of a seeded run. The graph shows the result of running a seeded setup. The Y axis is the total fitness score. The X axis is the generation number. The red line is the average fitness of the generation. The blue line is the top fitness of the generation.

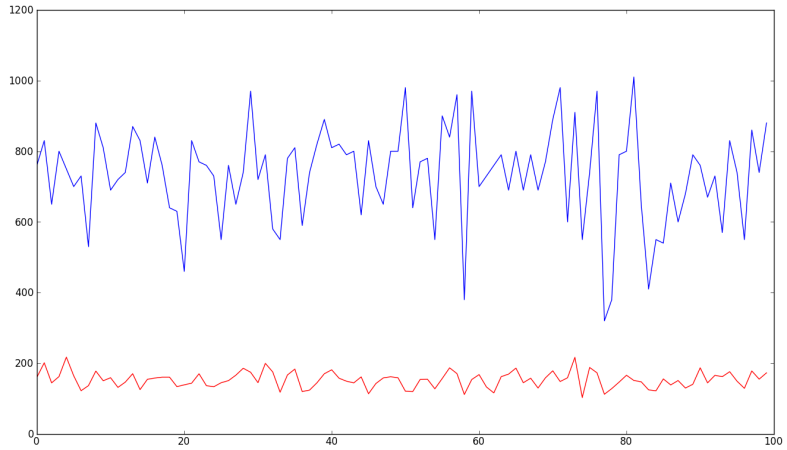

Figure 4.4: Fitness graph of an unseeded run. The graph shows the result of running an unseeded setup. The Y axis note the fitness value, and the X axis note the generations. The blue line is the best fitness and the red line is the average for the generation.

Best individual: $[21, 92, 85, 95]$

From the seeded run, the best individual has the values $[21, 92, 85, 95]$. Note that the highest score is 940, while the average is 106.80 in this run.

The resulting graph of the unseeded run can be seen in Figure 4.4. The differences is clear, as the unseeded top fitness varies greatly from generation to generation.

Unseeded - Last generation :

Generation 100, Average fitness 173.00 Top fitness 880.00

Best individual: $[45, 98, 97, 93]$

Both versions mostly agrees on the final value set through multiple runs. The unseeded version managed to hit a higher fitness than the seeded version on several occations:

[71][98][48][92] = Fitness: 1010.0

[30][95][90][94] = Fitness: 980.0

[7][95][24][95] = Fitness: 970.0

Additionally, the average fitness is generally higher on the unseeded version, which can stem from a weakness in the seed's number series in the seeded version.

None of the parameter values get higher fitness by being 0, so it is proven that all the rules are needed, but a tendency is that the 2nd and 3rd rule is more important than the 1st rule, and therefore often get high-end values in high fitness sets.

This initial experiment used only the first and simplest of fitness functions, which favoured rule sets that produced the most road segments. The following experiment apply two fitness functions, total length and average lot size. The weights used were 1 for length and 10 for average lot size, in attempt to favour creating lots. The result can be seen in Figure 4.7 and Figure 4.8

Through many test runs with having all fitness functions activated, the results are very spread for non-seeded runs. With seeded runs and evenly weighted fitness functions, the GA seems to favour the last rule. This may stem from the fact that once this rule fires at a frequency higher then the first two it will dominate a large the portion of the combined fitness function. This frequently results in road networks similar to that of Figure 4.7

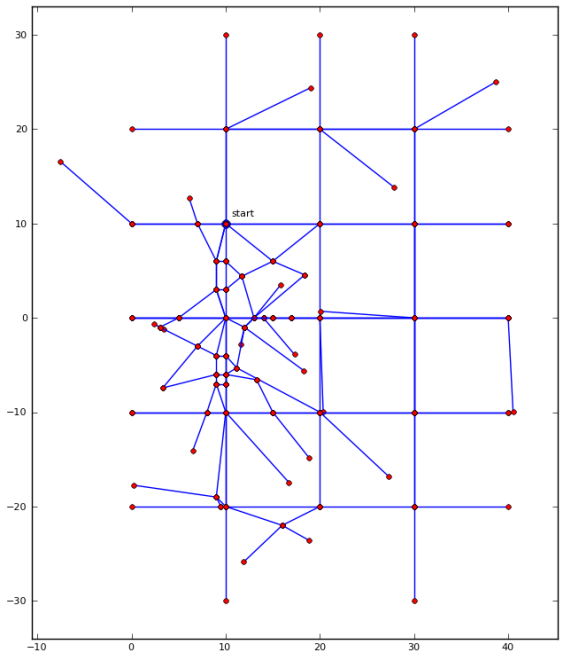

Figure 4.5: The resulting network by running a seeded setup.

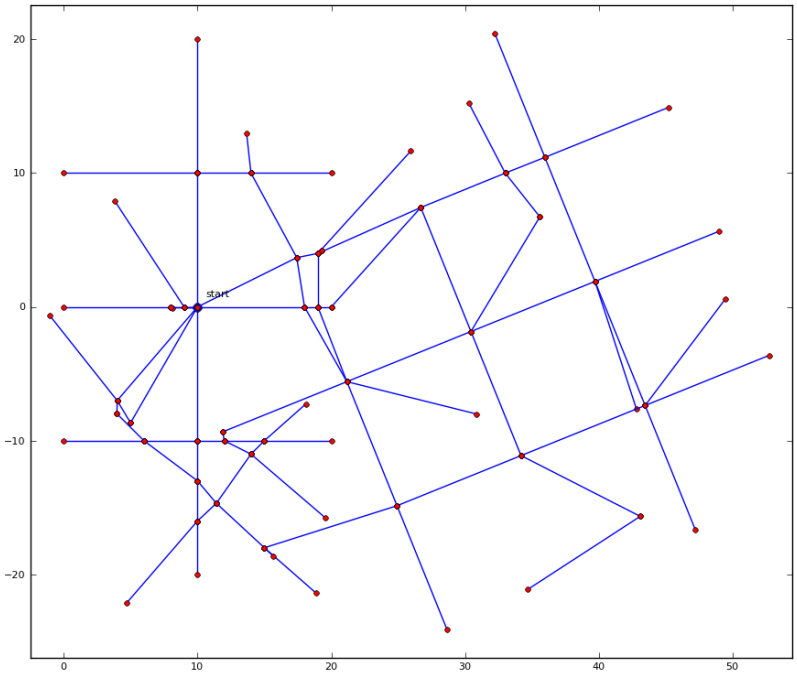

Figure 4.6: The resulting network by running a unseeded setup.

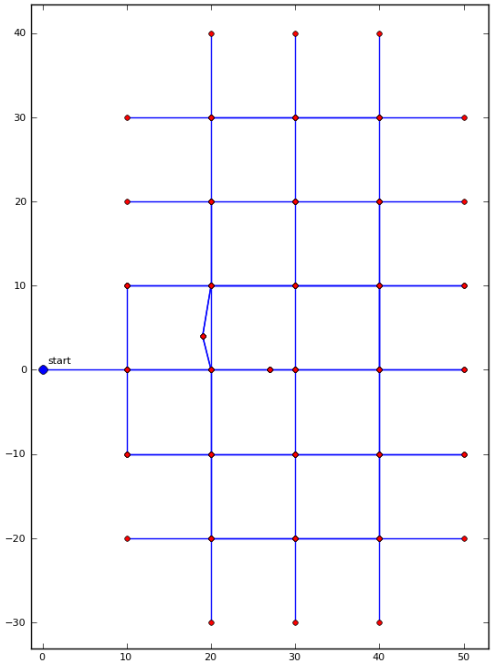

Figure 4.7: Network graph of a seeded run. The network is the result of running a seeded setup, with two fitness functions.

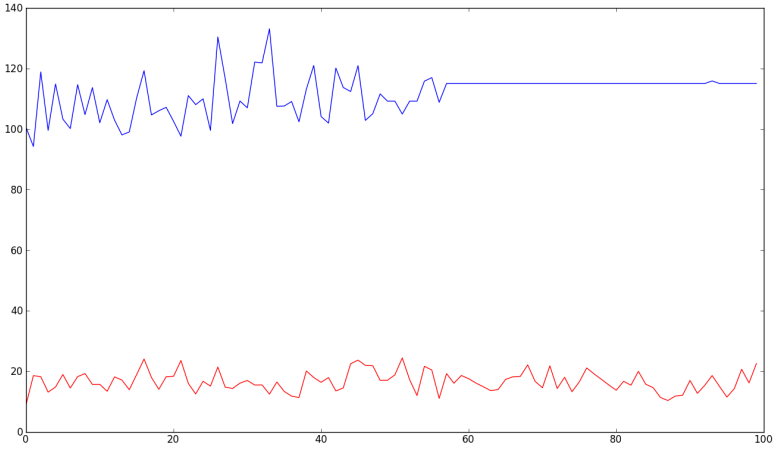

Figure 4.8: Fitness graph of an unseeded run. The graph shows the result of running a seeded setup, with two fitness functions. The Y axis note the fitness value, and the X axis note the generations. The blue line is the best fitness and the red line is the average for the generation.

4.3.3 Experiments Part 3 - GA + Flow

The following experiments use a predefined L-system from the previous experiments as an individual template for the GA individuals. The maximum flow algorithm determines the individuals fitness. Experiments show that in some networks, multiple optimal solutions may exists. In very large and complex networks there might be down to one single better solution.

The following experiment uses a previous L-system setup:

SP2L-system :

Axiom : F

Rule 1 : $F > () \rightarrow 0.6F()F(\#)$

Rule 2 : $F > F \rightarrow 0.5F()[F(\#)]$

Rule 3 : $F < [] > F \rightarrow 0.3[][F(\#)]$

This L-system generates the network seen in Figure 4.9. Seeding is used to get a consistent network.

GA :

Population size : 20

Generations : 20

Genome length : 10

Crossover rate : 90 %

Mutation rate : 10 %

Elitism : True

The GA converged immidiately on the best solution, as this is a simple network with only 10 intersections.

Best individual: [2][2][1][1][1][2][1][2][1][2] - Fitness = 29.00

The Figure 4.9 shows the result of the GA. Three sinks are used, but only two sinks are needed for a optimal result, as a bottleneck is close to the source.

A second run with the same GA, but with different L-system:

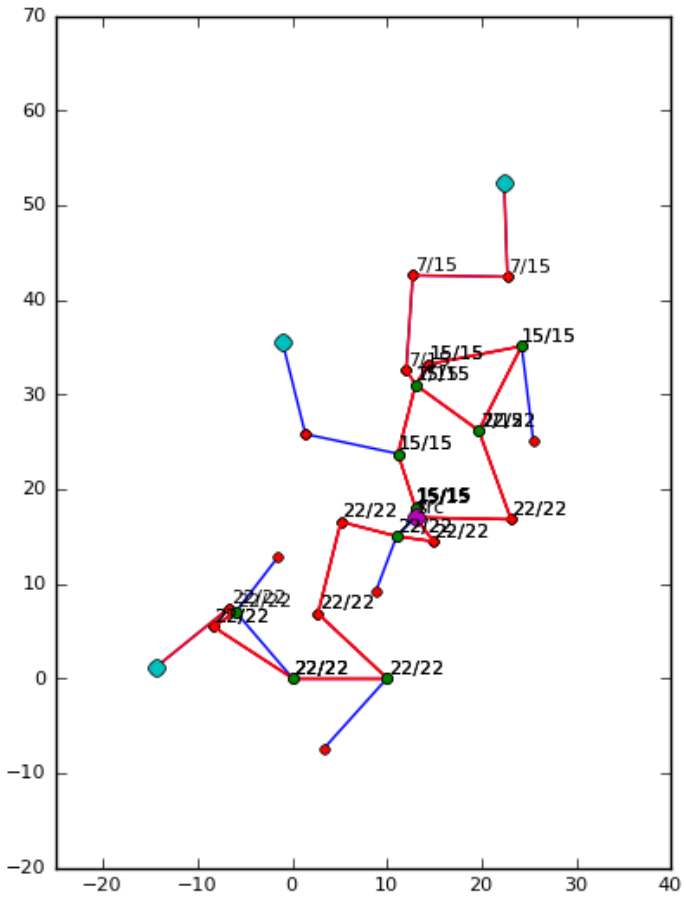

Figure 4.9: Figure shows a simple flow network result. The source is noted by the pink circle, and the sinks are noted by the light blue circles.The red path simulates the flow, with flow/capacity notations on the endpoints.

SP2L-system :

Axiom : F

Rule 1 : $F > () \rightarrow 1.0F()F(\#)$

Rule 2 : $F > F \rightarrow 0.6F()[F(\#)]$

Rule 2 : $> F \rightarrow 0.4F()X$

Rule 3 : $F < X > () \rightarrow 0.7[F(90)X][F(-90)X]F()$

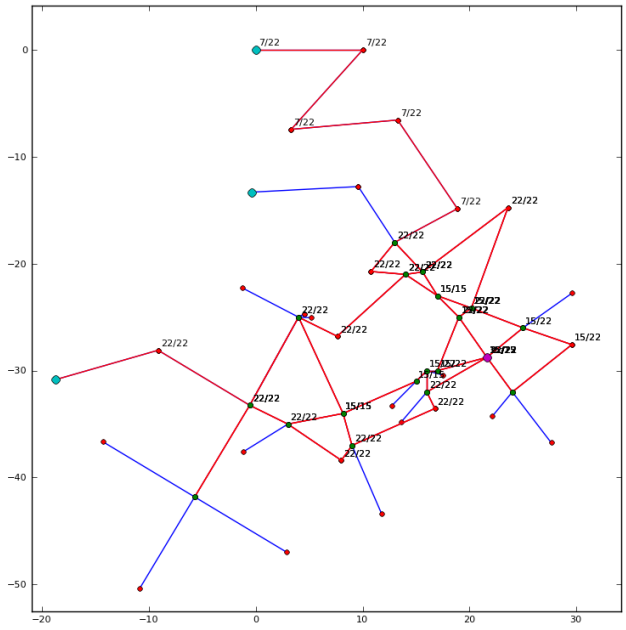

Figure 4.10: Figure shows a simple flow network result. The source is noted by the pink circle, and the sinks are noted by the light blue circles.The red path simulates the flow, with flow/capacity notations on the endpoints.

Best individual: [2][2][2][1][2][1][2][2][1][2][2][2][2][2][1][2][1][2][2] - Fitness = 29.00

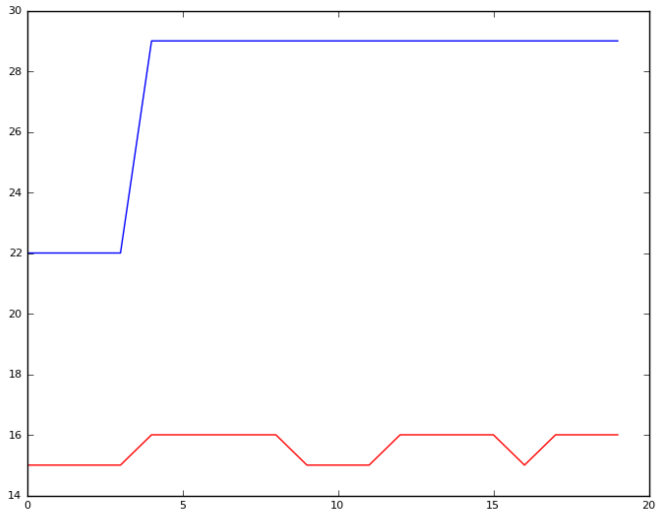

Figure 4.11: Figure shows a simple flow network GA result. The Y axis is the maximum flow, and the X axis is the generation.

Chapter 5

Evaluation and Conclusion

This chapter evaluates the work, answers the research questions and presents possible future work.

5.1 Evaluation

After a wide series of experiments and tuning, the experiments showed no real breakthrough results. The first phase, the L-system alone, shows good results when generating road networks, and yet there is much room for improvement. There is potential in L-systems and how local patterns in the road network can be arise without external functions, as opposed to the L-system in [PM01], that uses external functions such as successor correction.

The experiments run with the L-system and genetic algorithm combination showed it was difficult to find single optimal solutions without seeding, but the GA was often able to find a value range where it is inbetween more likely to get decent results.

Lastly the tuning of intersections helped map main road artheries into the network, which adds more a realism and surpasses [PM01]. It seems difficult to measure network flow inside the productions of L-systems, so the addition of intersection tuning proved to be a decent addition.

Following are conclusions and answers on the research questions posed at the beginning of this thesis:

Q : *What are the minimal and required features of an L-system to create valid road network models?*

A : From the experiments it could be observed that the rule sets needed to be stochastic, as to prevent unwanted patterns, but also include a chance to start a chain of more or less deterministic patterns areas.

Context sensitivity and parameters combined with a symbol module interpretation proved to be a strong feature.

Q : *Can an L-system be guided by a genetic algorithm in search of better automation of road network models?*

A : The research showed that it is possible, but theres is still much room for improvement and adaptation.

Q : *What fitness measures can guide the genetic algorithm?*

A : The research showed that a few fitness functions worked alone, though combining different fitness functions was more difficult and required alot of weight tuning.

Q : *Will a maximum flow-driven GA be able to tune flow transitions in the interpreted road network?*

A : Maximum flow driven GA proved to be able to tune intersections through a network.

5.2 Discussion

Due to the near endless possibilities in tuning L-systems and tweaking possibilities of genetic algorithms, the experiments had to be kept simple. Even with a simple L-system, good GA parameters were difficult to derive. The simple L-system might have been a key issue. Dividing an L-system into more distinct rules and parameters, might help a GA to easier distinguish between which rules are good or bad. An potential drawback with the L-systems used in the experiments is that they share a normalised probability that is widely affeted by the GA mutation and crossover. Additionally, the implementation of L-system functionality, its interpretation and the maximum flow support was complex and time-consuming. This contributed to the project barely touching the potential of these methods.

5.3 Future Work

With the discussion and limitations of the proposed system in mind, this section presents possible future work. The implemented system alone has already many extension possibilities and already implemented flexibility.

Possible extensions and ideas are listed in the following:

- The genetic algorithm and L-system combination can be explored further by adding series of parameters to the F() modules, and corresponding interpretation functionality. It would be interesting to see how a genetic algorithm would respond to higher degrees of tuning capabilities.

- A large expansion concerning the genetic algorithm, would be to see how a GA would be able to fully generate the production rules of an L-system from a pool of template rules.

- The last maximum flow phase may also be extended, by adding more intersection types such as yield intersections, where one or more directions have right of way, and the others must yield. The yielding directions get a flow penalty, which will allow the GA to find a flow path through the network similar to that of real motorways or right of way transport roads. This may require an additional graph layer where a set of an incoming lane, an outgoing lane and the inbetween intersection is combined into an imaginary intersection directly connected to the next intersection with holds a similar triplet. This is so that traffic from incoming ABC lanes, each only occupy 1/3 of the flow going through the outgoing lane D.

- Lastly the presented application must be extended with a parser to actually generate the 3D road network models from the object state in the application.

Bibliography

[DHL+98] Oliver Deussen, Pat Hanrahan, Bernd Lintermann, Radomír Měch, Matt Pharr, and Przemyslaw Prusinkiewicz. Realistic modeling and rendering of plant ecosystems. *Proceedings of the 25th annual conference on Computer graphics and interactive techniques, 275-286, ACM, New York, NY, USA*, 1998.

[FM08] David Floreano and Claudio Mattiussi. *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies.* The MIT Press, Cambridge, Massachusetts, USA, 2008.

[For56] Jr.; Fulkerson D.R. Ford, L.R. Maximal flow through a network. *Canadian Journal of Mathematics pp.399-404*, 1956.

[GH75] D. L. Gerlough and M. J. Huber. Traffic flow theory- a monograph. *Special Report 165, Transportation Research Board*, 1975.

[GR98] Andrew V. Goldberg and Satish Rao. Beyond the flow decomposition barrier. *J. ACM*, pages 783–797, 1998.

[Hol75] J. H. Holland. Adaptation in natural and artificial systems. *University of Michigan Press, Ann Arbor*, 1975.

[HP01] G. Hornby and J. Pollack. Evolving L-systems to generate virtual creatures. *Computers & graphics. 25(6):1041-1048*, 2001.

[Jac01] Christian Jacob. *Illustrating Evolutionary Computation with Mathematica.* Morgan Kaufmann, 2001.

[Lin68] Aristid Lindenmayer. Mathematical models for cellular interactions in development. *Journal of Theoretical Biology, 18(3):280-299*, 1968.

[Man06] Stelios Manousakis. Musical L-systems. *Master's Thesis, Institute of Sonology*, 2006.

[NSW03] Hansrudi Noser, Peter Stucki, and W. Wellauer. Rule-based animation system with gas as testbed for generic evolutionary applications. *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'02), November 18-22, 2002, Singapore, pp 355-359*, 2003.

[Och98] Gabriela Ochoa. On Genetic Algorithms and lindenmayer systems. In *PARALLEL PROBLEM SOLVING FROM NATURE V*, pages 335–344. Springer-Verlag, 1998.

[PHHM96] Przemyslaw Prusinkiewiczy, Mark Hammel, Jim Hanan, and Radomír Měch. Visual models of plant development. *Handbook of formal languages, vol. 3, Pages 535 - 597, Springer-Verlag New York, Inc. New York, NY, USA*, 1996.

[PJM94] Przemyslaw Prusinkiewicz, Mark James, and Radomír Měch. Synthetic topiary. In *SIGGRAPH '94 Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 351–358, New York, NY, USA, 1994. ACM.

[PL90] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. The Algorithmic beauty of plants. *Springer Verlag, Berlin*, 1990.

[PM01] Yoav I. H. Parish and Pascal Müller. Procedural city generation. *In Proceedings of ACM SIGGRAPH 2001, ACM Press / ACM SIGGRAPH, New York. E. Fiume (ed), COMPUTER GRAPHICS, Annual Conference Series, ACM, pages 301-308*, 2001.

[Pru86] Przemyslaw Prusinkiewicz. Graphical applications of L-systems. *In Proceedings on Graphics Interface '86, pages 247-253. Canadian Information Processing Society, Toronto*, 1986.

[RCB+02] Bian Runqiang, Yi-Ping Phoebe Chen, Kevin Burrage, Jim Hanan, Peter Room, and John Belward. Derivation of L-system models from measurements of biological branching structures using Genetic Algorithms. In *Proceedings of the 15th international conference on Industrial and engineering applications of artificial intelligence and expert systems: developments in applied artificial intelligence*, IEA/AIE '02, pages 514–524, London, UK, UK, 2002. Springer-Verlag.

[RLFS02] Yodthong Rodkaew, Chidchanok Lursinsap, Tadahiro Fujimoto, and Suchada Siripant. Modeling leaf shapes using L-systems

and Genetic Algorithms. In *In International Conference NICOGRAPH (April)*, pages 73–78, 2002.

[Veg08] Statens Vegvesen. *Statens Vegvesen Håndbok 017 Vei- og gateutforming*. www.vegvesen.no, 2008.