CITYBREEDER:
CITY DESIGN WITH EVOLUTIONARY COMPUTATION

by

Adam T.S. Cohen

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of

MASTER OF COMPUTER SCIENCE

School of Computer Science

at

CARLETON UNIVERSITY

Ottawa, Ontario
September, 2013

© Copyright by Adam T.S. Cohen, 2013
# Table of Contents

List of Tables viii  
List of Figures x  
List of Algorithms xviii  
Abstract xx  
Acknowledgements xxi  

## Chapter 1 Introduction  
1.1 Overview ................................................................. 1  
1.2 The Problem Statement .................................................. 2  
  1.2.1 Motivation ........................................................ 2  
  1.2.2 Solving the Problem .............................................. 3  
1.3 Scope ........................................................................... 3  
  1.3.1 Features .............................................................. 4  
  1.3.2 System Design and Implementation ............................... 4  
  1.3.3 Experimentation ..................................................... 4  
1.4 Definitions ................................................................. 5  
  1.4.1 Rapid ................................................................. 5  
  1.4.2 User-Guided ........................................................ 5  
  1.4.3 City Designs ........................................................ 6  
  1.4.4 Blending of Multiple Existing City Designs .................... 6  
  1.4.5 Derived from Real Cities ......................................... 6  
1.5 Contributions .................................................................. 6  
1.6 Thesis Organization ....................................................... 7
<table>
<thead>
<tr>
<th>Chapter 4</th>
<th>System Design and Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Symbol Tables</td>
</tr>
<tr>
<td>4.2</td>
<td>Pipeline</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Import</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Discovering City Genes</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Breeding Cities</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Evolutionary Computation and Genetic Programming</td>
</tr>
<tr>
<td>4.3</td>
<td>Representation</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Genotype</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Phenotype</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Expression Mechanism</td>
</tr>
<tr>
<td>4.4</td>
<td>Operators</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Crossover</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Mutation</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Selection</td>
</tr>
<tr>
<td>4.5</td>
<td>Fitness Evaluation</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Categorical Map Similarity</td>
</tr>
<tr>
<td>4.6</td>
<td>How the System Answers the Problem Statement</td>
</tr>
<tr>
<td>4.7</td>
<td>Summary</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 5</th>
<th>Expression Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>5.2</td>
<td>Genotype Layer Surface</td>
</tr>
<tr>
<td>5.3</td>
<td>Simple Growth</td>
</tr>
<tr>
<td>5.3.1</td>
<td>All Zeros</td>
</tr>
</tbody>
</table>
### 5.3.2 All Ones

Page 121

### 5.4 Complex Growth

Page 122

- **5.4.1 North West Density**  
  Page 123
- **5.4.2 South East Density**  
  Page 124
- **5.4.3 Central Density**  
  Page 125

### 5.5 Simple Rotation

Page 126

- **5.5.1 Simple Rotation 0**  
  Page 127
- **5.5.2 Simple Rotation 0.25**  
  Page 127
- **5.5.3 Simple Rotation 0.5**  
  Page 128
- **5.5.4 Simple Rotation 0.75**  
  Page 129
- **5.5.5 Simple Rotation 1**  
  Page 130

### 5.6 Complex Rotation and Angle Contiguity Values

Page 131

- **5.6.1 North East Rotated, Angle Contiguity 1**  
  Page 132
- **5.6.2 North East Rotated, Angle Contiguity 3**  
  Page 133
- **5.6.3 North East Rotated, Angle Contiguity 5**  
  Page 133
- **5.6.4 North East Rotated, Angle Contiguity 50**  
  Page 134

### 5.7 Summary

Page 134

### Chapter 6 Simple Evolution

Page 135

- **6.1 Introduction**  
  Page 135
- **6.2 Simple Targets**  
  Page 135
  - **6.2.1 Experiment 1**  
    Page 137
  - **6.2.2 Experiment 2**  
    Page 144
  - **6.2.3 Experiment 3**  
    Page 150
- **6.3 Individual Operators**  
  Page 159
  - **6.3.1 Basic Crossover**  
    Page 160
  - **6.3.2 Weak Context-Preserving Crossover**  
    Page 162
  - **6.3.3 Expand Leaf to Quadtree Mutation**  
    Page 165
  - **6.3.4 Compress Leaves-Only Quadtree to Leaf Mutation**  
    Page 168
  - **6.3.5 Gaussian Mutate Leaf Value Mutation**  
    Page 171

---

v
List of Tables

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Ratings for <em>CityEngine</em></td>
</tr>
<tr>
<td>3.2</td>
<td>Ratings for <em>Image Analogies</em></td>
</tr>
<tr>
<td>3.3</td>
<td>Ratings for <em>Interactive Procedural Street Modeling</em></td>
</tr>
<tr>
<td>3.4</td>
<td>Ratings for <em>Interactive Geometric Simulation of 4D Cities</em></td>
</tr>
<tr>
<td>3.5</td>
<td>Ratings for <em>Procedural City Layout Generation Based on Urban Land Use Models</em></td>
</tr>
<tr>
<td>3.6</td>
<td>Ratings for <em>Interactive Design of Urban Spaces Using Geometrical and Behavioral Modeling</em></td>
</tr>
<tr>
<td>3.7</td>
<td>Ratings for <em>Generating settlement structures: a method for urban planning and analysis supported by cellular automata</em></td>
</tr>
<tr>
<td>3.8</td>
<td>Ratings for <em>Interactive Modeling of City Layouts using Layers of Procedural Content</em></td>
</tr>
<tr>
<td>3.9</td>
<td>Ratings for <em>Interactive Example-Based Urban Layout Synthesis</em></td>
</tr>
<tr>
<td>3.10</td>
<td>Ratings for <em>City Generator: GIS Driven Evolution in Urban Simulation</em></td>
</tr>
<tr>
<td>3.11</td>
<td>Ratings for <em>Terrain generation using an Interactive Genetic Algorithm</em></td>
</tr>
<tr>
<td>3.12</td>
<td>Ratings Summary</td>
</tr>
<tr>
<td>4.1</td>
<td>Fixed Values Symbol Table</td>
</tr>
<tr>
<td>4.2</td>
<td>Parameters Symbol Table</td>
</tr>
<tr>
<td>6.1</td>
<td>Simple Evolution Experiment Settings</td>
</tr>
<tr>
<td>6.2</td>
<td>Summary of Individual Operators Simple Evolution Results</td>
</tr>
<tr>
<td>7.1</td>
<td>Ottawa Useable Coordinates</td>
</tr>
<tr>
<td>7.2</td>
<td>Ottawa Exported Coordinates</td>
</tr>
<tr>
<td>7.3</td>
<td>Discovering City Genes Ottawa Experiment Settings</td>
</tr>
<tr>
<td>7.4</td>
<td>Ottawa Similarity Values</td>
</tr>
<tr>
<td>7.5</td>
<td>New York Useable Coordinates</td>
</tr>
<tr>
<td>7.6</td>
<td>New York Exported Coordinates</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Crossover Illustration Parents</td>
<td>21</td>
</tr>
<tr>
<td>2.2</td>
<td>Valid Crossover Swap Points in Parent B</td>
<td>22</td>
</tr>
<tr>
<td>2.3</td>
<td>Permutation Mutation Illustration</td>
<td>24</td>
</tr>
<tr>
<td>2.4</td>
<td>Gaussian Mutation Illustration</td>
<td>25</td>
</tr>
<tr>
<td>3.1</td>
<td>Conceptual pipeline for procedural approaches to city design</td>
<td>39</td>
</tr>
<tr>
<td>3.2</td>
<td>Conceptual pipeline presented in <em>Modelling the Appearance and Behaviour of Urban Spaces</em> for procedural approaches to city design</td>
<td>40</td>
</tr>
<tr>
<td>3.3</td>
<td>Pipeline presented in <em>Procedural Modeling of Cities</em></td>
<td>41</td>
</tr>
<tr>
<td>3.4</td>
<td><em>CityEngine</em> mapped onto the conceptual model of procedural approaches to city design</td>
<td>43</td>
</tr>
<tr>
<td>3.5</td>
<td><em>Image Analogies</em> mapped onto the conceptual model of procedural approaches to city design</td>
<td>45</td>
</tr>
<tr>
<td>3.6</td>
<td>Pipeline presented in <em>Interactive Procedural Street Modeling</em></td>
<td>47</td>
</tr>
<tr>
<td>3.7</td>
<td><em>Interactive Procedural Street Modeling</em> mapped onto the conceptual model of procedural approaches to city design</td>
<td>48</td>
</tr>
<tr>
<td>3.8</td>
<td>Pipeline presented in <em>Interactive Geometric Simulation of 4D Cities</em></td>
<td>50</td>
</tr>
<tr>
<td>3.9</td>
<td><em>Interactive Geometric Simulation of 4D Cities</em> mapped onto the conceptual model of procedural approaches to city design</td>
<td>51</td>
</tr>
<tr>
<td>3.10</td>
<td><em>Procedural City Layout Generation Based on Urban Land Use Models</em> mapped onto the conceptual model of procedural approaches to city design</td>
<td>53</td>
</tr>
<tr>
<td>3.11</td>
<td>Pipeline presented in <em>Interactive Design of Urban Spaces Using Geometrical and Behavioral Modeling</em></td>
<td>54</td>
</tr>
<tr>
<td>3.12</td>
<td><em>Interactive Design of Urban Spaces Using Geometrical and Behavioral Modeling</em> mapped onto the conceptual model of procedural approaches to city design</td>
<td>55</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>4.13</td>
<td>Zoom Amount</td>
<td>103</td>
</tr>
<tr>
<td>4.14</td>
<td>Interactive Selection Window Displayed to User</td>
<td>108</td>
</tr>
<tr>
<td>4.15</td>
<td>Example Categorical Maps</td>
<td>111</td>
</tr>
<tr>
<td>4.16</td>
<td>Example Difference Images</td>
<td>112</td>
</tr>
<tr>
<td>5.1</td>
<td>All Zeroes Genotype: Depth 0</td>
<td>120</td>
</tr>
<tr>
<td>5.2</td>
<td>All Zeroes Expression: Ottawa Scale</td>
<td>120</td>
</tr>
<tr>
<td>5.3</td>
<td>All Zeroes Expression: Half Ottawa Scale</td>
<td>121</td>
</tr>
<tr>
<td>5.4</td>
<td>All Ones Genotype: Depth 0</td>
<td>121</td>
</tr>
<tr>
<td>5.5</td>
<td>All Ones Growth Expression: Ottawa Scale</td>
<td>122</td>
</tr>
<tr>
<td>5.6</td>
<td>All Ones Growth Expression: Half Ottawa Scale</td>
<td>122</td>
</tr>
<tr>
<td>5.7</td>
<td>North West Density Genotype</td>
<td>123</td>
</tr>
<tr>
<td>5.8</td>
<td>North West Density Growth Expression</td>
<td>123</td>
</tr>
<tr>
<td>5.9</td>
<td>South East Density Genotype</td>
<td>124</td>
</tr>
<tr>
<td>5.10</td>
<td>South East Density Growth Expression</td>
<td>125</td>
</tr>
<tr>
<td>5.11</td>
<td>Central Density Genotype</td>
<td>126</td>
</tr>
<tr>
<td>5.12</td>
<td>Central Density Growth Expression</td>
<td>126</td>
</tr>
<tr>
<td>5.13</td>
<td>Simple Rotation 0 Genotype</td>
<td>127</td>
</tr>
<tr>
<td>5.14</td>
<td>Simple Rotation 0 Expression</td>
<td>127</td>
</tr>
<tr>
<td>5.15</td>
<td>Simple Rotation 0.25 Genotype</td>
<td>128</td>
</tr>
<tr>
<td>5.16</td>
<td>Simple Rotation 0.25 Expression</td>
<td>128</td>
</tr>
<tr>
<td>5.17</td>
<td>Simple Rotation 0.5 Genotype</td>
<td>128</td>
</tr>
<tr>
<td>5.18</td>
<td>Simple Rotation 0.5 Expression</td>
<td>129</td>
</tr>
<tr>
<td>5.19</td>
<td>Simple Rotation 0.75 Genotype</td>
<td>129</td>
</tr>
<tr>
<td>5.20</td>
<td>Simple Rotation 0.75 Expression</td>
<td>130</td>
</tr>
<tr>
<td>5.21</td>
<td>Simple Rotation 1 Genotype</td>
<td>130</td>
</tr>
<tr>
<td>5.22</td>
<td>Simple Rotation 1 Expression</td>
<td>131</td>
</tr>
<tr>
<td>5.23</td>
<td>Complex Rotation Genotype, North East Rotated</td>
<td>132</td>
</tr>
<tr>
<td>5.24</td>
<td>North East Rotated, Angle Contiguity 1 Expression</td>
<td>132</td>
</tr>
<tr>
<td>5.25</td>
<td>North East Rotated, Angle Contiguity 3 Expression</td>
<td>133</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>5.26</td>
<td>North East Rotated, Angle Contiguity 5 Expression</td>
<td>133</td>
</tr>
<tr>
<td>5.27</td>
<td>North East Rotated, Angle Contiguity 50 Expression</td>
<td>134</td>
</tr>
<tr>
<td>6.1</td>
<td>Simple Experiment 1 Source Genotype</td>
<td>137</td>
</tr>
<tr>
<td>6.2</td>
<td>Simple Experiment 1 Target Phenotype</td>
<td>138</td>
</tr>
<tr>
<td>6.3</td>
<td>Evolution Fitness for Simple Example 1</td>
<td>139</td>
</tr>
<tr>
<td>6.4</td>
<td>Target and Evolved Simple Experiment 1</td>
<td>140</td>
</tr>
<tr>
<td>6.5</td>
<td>Composite of Evolved Phenotypes for Simple Example 1</td>
<td>141</td>
</tr>
<tr>
<td>6.6</td>
<td>Simple Experiment 1 Run 1 Genotype</td>
<td>142</td>
</tr>
<tr>
<td>6.7</td>
<td>Simple Experiment 1 Run 2 Genotype</td>
<td>142</td>
</tr>
<tr>
<td>6.8</td>
<td>Simple Experiment 1 Run 3 Genotype</td>
<td>143</td>
</tr>
<tr>
<td>6.9</td>
<td>Simple Experiment 1 Run 4 Genotype</td>
<td>143</td>
</tr>
<tr>
<td>6.10</td>
<td>Simple Experiment 1 Run 5 Genotype</td>
<td>143</td>
</tr>
<tr>
<td>6.11</td>
<td>Simple Experiment 2 Source Genotype</td>
<td>144</td>
</tr>
<tr>
<td>6.12</td>
<td>Simple Experiment 2 Target Phenotype</td>
<td>145</td>
</tr>
<tr>
<td>6.13</td>
<td>Evolution Fitness for Simple Example 2</td>
<td>146</td>
</tr>
<tr>
<td>6.14</td>
<td>Target and Evolved Simple Experiment 2</td>
<td>147</td>
</tr>
<tr>
<td>6.15</td>
<td>Composite of Evolved Phenotypes for Simple Example 2</td>
<td>148</td>
</tr>
<tr>
<td>6.16</td>
<td>Simple Experiment 2 Run 1 Genotype</td>
<td>149</td>
</tr>
<tr>
<td>6.17</td>
<td>Simple Experiment 2 Run 2 Genotype</td>
<td>149</td>
</tr>
<tr>
<td>6.18</td>
<td>Simple Experiment 2 Run 3 Genotype</td>
<td>149</td>
</tr>
<tr>
<td>6.19</td>
<td>Simple Experiment 2 Run 4 Genotype</td>
<td>150</td>
</tr>
<tr>
<td>6.20</td>
<td>Simple Experiment 2 Run 5 Genotype</td>
<td>150</td>
</tr>
<tr>
<td>6.21</td>
<td>Simple Experiment 3 Source Genotype</td>
<td>151</td>
</tr>
<tr>
<td>6.22</td>
<td>Simple Experiment 3 Target Phenotype</td>
<td>151</td>
</tr>
<tr>
<td>6.23</td>
<td>Evolution Fitness for Simple Example 3</td>
<td>152</td>
</tr>
<tr>
<td>6.24</td>
<td>Simple Experiment 3 Run 1 Phenotype. Fitness 0.98268</td>
<td>153</td>
</tr>
<tr>
<td>6.25</td>
<td>Simple Experiment 3 Run 1 Genotype</td>
<td>153</td>
</tr>
<tr>
<td>6.26</td>
<td>Simple Experiment 3 Run 2 Phenotype. Fitness 0.99214</td>
<td>154</td>
</tr>
<tr>
<td>6.27</td>
<td>Simple Experiment 3 Run 2 Genotype</td>
<td>154</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>6.28</td>
<td>Simple Experiment 3 Run 3 Phenotype. Fitness 0.98269</td>
<td></td>
</tr>
<tr>
<td>6.29</td>
<td>Simple Experiment 3 Run 3 Genotype</td>
<td></td>
</tr>
<tr>
<td>6.30</td>
<td>Simple Experiment 3 Run 4 Phenotype. Fitness 0.985560</td>
<td></td>
</tr>
<tr>
<td>6.31</td>
<td>Simple Experiment 3 Run 4 Genotype</td>
<td></td>
</tr>
<tr>
<td>6.32</td>
<td>Simple Experiment 3 Run 5 Phenotype. Fitness 0.98903</td>
<td></td>
</tr>
<tr>
<td>6.33</td>
<td>Simple Experiment 3 Run 5 Genotype</td>
<td></td>
</tr>
<tr>
<td>6.34</td>
<td>Composite of Evolved Phenotypes for Simple Example 3</td>
<td></td>
</tr>
<tr>
<td>6.35</td>
<td>Target and Evolved Simple Experiment 3</td>
<td></td>
</tr>
<tr>
<td>6.36</td>
<td>Basic Crossover Fitness</td>
<td></td>
</tr>
<tr>
<td>6.37</td>
<td>Basic Crossover Only Evolved Phenotype</td>
<td></td>
</tr>
<tr>
<td>6.38</td>
<td>Basic Crossover Only Evolved Genotype</td>
<td></td>
</tr>
<tr>
<td>6.39</td>
<td>Basic Crossover Only Evolved Genotype (RGB)</td>
<td></td>
</tr>
<tr>
<td>6.40</td>
<td>Weak Context-Preserving Crossover Fitness</td>
<td></td>
</tr>
<tr>
<td>6.41</td>
<td>Weak Context-Preserving Crossover Only Evolved Phenotype</td>
<td></td>
</tr>
<tr>
<td>6.42</td>
<td>Weak Context-Preserving Crossover Only Evolved Genotype</td>
<td></td>
</tr>
<tr>
<td>6.43</td>
<td>Weak Context-Preserving Crossover Only Evolved Genotype (RGB)</td>
<td></td>
</tr>
<tr>
<td>6.44</td>
<td>Expand Leaf to Quadtree Mutation Fitness</td>
<td></td>
</tr>
<tr>
<td>6.45</td>
<td>Expand Leaf to Quadtree Mutation Only Evolved Phenotype</td>
<td></td>
</tr>
<tr>
<td>6.46</td>
<td>Expand Leaf to Quadtree Mutation Only Evolved Genotype</td>
<td></td>
</tr>
<tr>
<td>6.47</td>
<td>Expand Leaf to Quadtree Mutation Only Evolved Genotype (RGB)</td>
<td></td>
</tr>
<tr>
<td>6.48</td>
<td>Compress Leaves-Only Quadtree to Leaf Mutation Fitness</td>
<td></td>
</tr>
<tr>
<td>6.49</td>
<td>Compress Leaves-Only Quadtree to Leaf Mutation Only Evolved Phenotype</td>
<td></td>
</tr>
<tr>
<td>6.50</td>
<td>Compress Leaves-Only Quadtree to Leaf Mutation Only Evolved Genotype</td>
<td></td>
</tr>
<tr>
<td>6.51</td>
<td>Compress Leaves-Only Quadtree to Leaf Mutation Only Evolved Genotype (RGB)</td>
<td></td>
</tr>
<tr>
<td>6.52</td>
<td>Gaussian Mutate Leaf Value Mutation Fitness</td>
<td></td>
</tr>
<tr>
<td>6.53</td>
<td>Gaussian Mutate Leaf Value Mutation Only Evolved Phenotype</td>
<td></td>
</tr>
<tr>
<td>6.54</td>
<td>Gaussian Mutate Leaf Value Mutation Only Evolved Genotype</td>
<td></td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>6.55</td>
<td>Gaussian Mutate Leaf Value Mutation Only Evolved Genotype (RGB)</td>
<td>174</td>
</tr>
<tr>
<td>6.56</td>
<td>Permute Leaves-Only Quadtree Subtrees Mutation Fitness</td>
<td>175</td>
</tr>
<tr>
<td>6.57</td>
<td>Permute Leaves-Only Quadtree Subtrees Mutation Only Evolved Phenotype</td>
<td>176</td>
</tr>
<tr>
<td>6.58</td>
<td>Permute Leaves-Only Quadtree Subtrees Mutation Only Evolved Genotype</td>
<td>176</td>
</tr>
<tr>
<td>6.59</td>
<td>Permute Leaves-Only Quadtree Subtrees Mutation Only Evolved Genotype (RGB)</td>
<td>177</td>
</tr>
<tr>
<td>6.60</td>
<td>Permute Quadtree Subtrees Mutation Fitness</td>
<td>178</td>
</tr>
<tr>
<td>6.61</td>
<td>Permute Quadtree Subtrees Mutation Only Evolved Phenotype</td>
<td>179</td>
</tr>
<tr>
<td>6.62</td>
<td>Permute Quadtree Subtrees Mutation Only Evolved Genotype</td>
<td>179</td>
</tr>
<tr>
<td>6.63</td>
<td>Permute Quadtree Subtrees Mutation Only Evolved Genotype (RGB)</td>
<td>180</td>
</tr>
<tr>
<td>7.1</td>
<td>OpenStreetMap Screenshot for Ottawa with Useable AreaOutlined</td>
<td>184</td>
</tr>
<tr>
<td>7.2</td>
<td>OpenStreetMap Screenshot for Ottawa with Exported AreaOutlined</td>
<td>184</td>
</tr>
<tr>
<td>7.3</td>
<td>Ottawa Imported (Target) Phenotype</td>
<td>185</td>
</tr>
<tr>
<td>7.4</td>
<td>Ottawa Imported (Target) Phenotype 1/4 Zoom</td>
<td>186</td>
</tr>
<tr>
<td>7.5</td>
<td>Ottawa Categorical Maps</td>
<td>187</td>
</tr>
<tr>
<td>7.6</td>
<td>Evolution Fitness for Ottawa</td>
<td>189</td>
</tr>
<tr>
<td>7.7</td>
<td>Ottawa Run 1 Phenotype</td>
<td>190</td>
</tr>
<tr>
<td>7.8</td>
<td>Ottawa Run 1 Genotype</td>
<td>191</td>
</tr>
<tr>
<td>7.9</td>
<td>Target and Evolved Ottawa</td>
<td>192</td>
</tr>
<tr>
<td>7.10</td>
<td>Evolved Ottawa Categorical Maps</td>
<td>193</td>
</tr>
<tr>
<td>7.11</td>
<td>Ottawa Difference Images</td>
<td>194</td>
</tr>
<tr>
<td>7.12</td>
<td>Ottawa Run 3 Phenotype</td>
<td>195</td>
</tr>
<tr>
<td>7.13</td>
<td>OpenStreetMap Screenshot for New York with Useable AreaOutlined</td>
<td>197</td>
</tr>
<tr>
<td>7.14</td>
<td>OpenStreetMap Screenshot for New York with Exported AreaOutlined</td>
<td>197</td>
</tr>
<tr>
<td>7.15</td>
<td>New York Imported (Target) Phenotype</td>
<td>199</td>
</tr>
<tr>
<td>7.16</td>
<td>New York Imported (Target) Phenotype 1/7 Zoom</td>
<td>200</td>
</tr>
<tr>
<td>7.17</td>
<td>New York Categorical Maps</td>
<td>201</td>
</tr>
</tbody>
</table>

xv
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.18</td>
<td>Evolution Fitness for New York</td>
<td>202</td>
</tr>
<tr>
<td>7.19</td>
<td>New York Run 4 Phenotype</td>
<td>203</td>
</tr>
<tr>
<td>7.20</td>
<td>New York Run 4 Genotype</td>
<td>203</td>
</tr>
<tr>
<td>7.21</td>
<td>Target and Evolved New York</td>
<td>204</td>
</tr>
<tr>
<td>7.22</td>
<td>Evolved New York Categorical Maps</td>
<td>205</td>
</tr>
<tr>
<td>7.23</td>
<td>New York Difference Images</td>
<td>206</td>
</tr>
<tr>
<td>7.24</td>
<td><em>OpenStreetMap</em> Screenshot for Paris with Useable Area Outlined</td>
<td>208</td>
</tr>
<tr>
<td>7.25</td>
<td><em>OpenStreetMap</em> Screenshot for Paris with Exported Area Outlined</td>
<td>208</td>
</tr>
<tr>
<td>7.26</td>
<td>Paris Imported (Target) Phenotype</td>
<td>209</td>
</tr>
<tr>
<td>7.27</td>
<td>Paris Imported (Target) Phenotype 1/3 Zoom</td>
<td>210</td>
</tr>
<tr>
<td>7.28</td>
<td>Paris Categorical Maps</td>
<td>211</td>
</tr>
<tr>
<td>7.29</td>
<td>Evolution Fitness for Paris with Angle Contiguity 10</td>
<td>212</td>
</tr>
<tr>
<td>7.30</td>
<td>Evolution Fitness for Paris with Angle Contiguity 1</td>
<td>213</td>
</tr>
<tr>
<td>7.31</td>
<td>Paris Angle Contiguity 10 Run 5 Phenotype</td>
<td>214</td>
</tr>
<tr>
<td>7.32</td>
<td>Paris Angle Contiguity 10 Run 5 Genotype</td>
<td>214</td>
</tr>
<tr>
<td>7.33</td>
<td>Paris Angle Contiguity 1 Run 2 Phenotype</td>
<td>215</td>
</tr>
<tr>
<td>7.34</td>
<td>Paris Angle Contiguity 1 Run 2 Genotype</td>
<td>215</td>
</tr>
<tr>
<td>7.35</td>
<td>Target and Evolved Paris</td>
<td>216</td>
</tr>
<tr>
<td>7.36</td>
<td>Evolved Paris Categorical Maps</td>
<td>217</td>
</tr>
<tr>
<td>7.37</td>
<td>Paris Difference Images</td>
<td>218</td>
</tr>
<tr>
<td>8.1</td>
<td>Breeding Cities User Selection, Dense Demo, Generation 1</td>
<td>222</td>
</tr>
<tr>
<td>8.2</td>
<td>Dense Target Demonstration, Generation 1 Individual</td>
<td>223</td>
</tr>
<tr>
<td>8.3</td>
<td>Breeding Cities User Selection, Dense Demo, Generation 2</td>
<td>223</td>
</tr>
<tr>
<td>8.4</td>
<td>Breeding Cities User Selection, Dense Demo, Generation 3</td>
<td>224</td>
</tr>
<tr>
<td>8.5</td>
<td>Breeding Cities User Selection, Dense Demo, Generation 4</td>
<td>224</td>
</tr>
<tr>
<td>8.6</td>
<td>Breeding Cities User Selection, Dense Demo, Generation 5</td>
<td>225</td>
</tr>
<tr>
<td>8.7</td>
<td>Dense Target Demonstration, Generation 5 Individual</td>
<td>226</td>
</tr>
<tr>
<td>8.8</td>
<td>Breeding Cities User Selection, Sparse Demo, Generation 1</td>
<td>226</td>
</tr>
<tr>
<td>8.9</td>
<td>Breeding Cities User Selection, Sparse Demo, Generation 2</td>
<td>227</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>8.10</td>
<td>Breeding Cities User Selection, Sparse Demo, Generation 3</td>
<td>228</td>
</tr>
<tr>
<td>8.11</td>
<td>Breeding Cities User Selection, Sparse Demo, Generation 4</td>
<td>228</td>
</tr>
<tr>
<td>8.12</td>
<td>Breeding Cities User Selection, Sparse Demo, Generation 5</td>
<td>229</td>
</tr>
<tr>
<td>8.13</td>
<td>Sparse Target Demonstration, Generation 5 Individual</td>
<td>230</td>
</tr>
<tr>
<td>8.14</td>
<td>Ottawa Genotype with Crossover Points Highlighted</td>
<td>231</td>
</tr>
<tr>
<td>8.15</td>
<td>New York Genotype with Crossover Points Highlighted</td>
<td>231</td>
</tr>
<tr>
<td>8.16</td>
<td>City Breeding Example, Scale X Crossover, Individual 1</td>
<td>232</td>
</tr>
<tr>
<td>8.17</td>
<td>City Breeding Example, Scale X Crossover, Individual 2</td>
<td>233</td>
</tr>
<tr>
<td>8.18</td>
<td>City Breeding Example, Scale Y Crossover, Individual 1</td>
<td>234</td>
</tr>
<tr>
<td>8.19</td>
<td>City Breeding Example, Scale Y Crossover, Individual 2</td>
<td>234</td>
</tr>
<tr>
<td>8.20</td>
<td>City Breeding Example, Angle Crossover, Individual 1</td>
<td>235</td>
</tr>
<tr>
<td>8.21</td>
<td>City Breeding Example, Angle Crossover, Individual 2</td>
<td>235</td>
</tr>
<tr>
<td>8.22</td>
<td>City Breeding Example, Complete Crossover, Individual 1</td>
<td>236</td>
</tr>
<tr>
<td>8.23</td>
<td>City Breeding Example, Complete Crossover, Individual 2</td>
<td>236</td>
</tr>
</tbody>
</table>
List of Algorithms

1. Evolutionary Computation Pipeline .................................................. 16
2. Ramped-Half-and-Half Initialization .................................................. 18
3. Full Initialization ........................................................................... 18
4. Grow Initialization .......................................................................... 19
5. Basic Crossover .............................................................................. 20
6. Weak Context-Preserving Crossover ............................................... 23
7. Rank Selection ................................................................................ 26
8. Preparation for Rank Selection ....................................................... 27
9. Uniform Selection .......................................................................... 27
10. Extract Regions ............................................................................ 31
11. Find All Wedges ........................................................................... 32
12. Group Wedges into Regions ........................................................... 33
13. Rotate Point .................................................................................. 34
14. Discovering City Genes Mapped onto the Evolutionary Computation Pipeline 79
15. Breeding Cities Mapped onto the Evolutionary Computation Pipeline 79
16. Prepopulated Initialization .............................................................. 92
17. Genotype-to-Phenotype Expression ................................................ 95
18. Grow (Expression) ......................................................................... 96
19. Lookup Growth Extension .............................................................. 97
20. Lookup Average Value ................................................................... 97
21. Add Proposed Area ......................................................................... 97
22. Rotate ............................................................................................. 98
23. Create Angular Enclosure Histograms ............................................ 98
24. Get Rotation Layers ....................................................................... 99
25. Apply Rotation ............................................................................... 99
26. Combine Rotation Layers ............................................................... 100
27. Identify Enclosure Angles ............................................................... 100
28. Zoom ............................................................................................. 101
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>Crossover</td>
<td>104</td>
</tr>
<tr>
<td>30</td>
<td>Mutation</td>
<td>105</td>
</tr>
<tr>
<td>31</td>
<td>Expand Leaf to Quadtree Mutation</td>
<td>105</td>
</tr>
<tr>
<td>32</td>
<td>Compress Leaves-Only Quadtree to Leaf Mutation</td>
<td>106</td>
</tr>
<tr>
<td>33</td>
<td>Gaussian Mutate Leaf Value Mutation</td>
<td>106</td>
</tr>
<tr>
<td>34</td>
<td>Permute Leaves-Only Quadtree Subtrees Mutation</td>
<td>107</td>
</tr>
<tr>
<td>35</td>
<td>Permute Quadtree Subtrees Mutation</td>
<td>107</td>
</tr>
<tr>
<td>36</td>
<td>Fitness Function</td>
<td>113</td>
</tr>
<tr>
<td>37</td>
<td>Find Enclosures</td>
<td>113</td>
</tr>
<tr>
<td>38</td>
<td>Generate Categorical Maps</td>
<td>114</td>
</tr>
<tr>
<td>39</td>
<td>Categorical Maps Fitness</td>
<td>115</td>
</tr>
<tr>
<td>40</td>
<td>Calculate Weighted Similarity</td>
<td>116</td>
</tr>
</tbody>
</table>
Abstract

Cities are complex entities which play important roles in the lives of the many people who inhabit them. The process of creating city designs is a complex, time-consuming endeavour, pursued by several different groups. Though procedural techniques have been developed to speed up this process, virtually none enable the creation of designs based on multiple existing designs. This thesis presents CityBreeder, a system which enables the rapid, user-guided development of city designs based on the blending of multiple existing city designs. Almost no previous research has been conducted regarding this capability.

This capability is achieved through the use of Evolutionary Computation, which is used to discover the genetic representation of existing city designs derived from real city data obtained from OpenStreetMap. Once discovered, these cities can be ‘bred’ together, creating new offspring designs. More of this thesis is concerned with the first portion of this task: the discovery of the genetic representations of real city designs. The combination of these cities is given less attention, but is explored through several demonstrations which show this capability is achieved.

More specifically, CityBreeder employs Genetic Programming on a layered quadtree genotype representation to create phenotype city designs consisting of road layouts comprised of nodes and edges. Additionally, a genotype-to-phenotype expression mechanism, genetic operators and a fitness function employing computational geometry techniques are presented and tested, all of which are tailored to the city design context. Experiments and examples are shown which analyze the system’s representation and operators using simple, artificially constructed data, as well as through experiments showing the system functioning with data derived from real cities.
Acknowledgements

I would like to thank several people for their support, which was invaluable during the time I spent working on this thesis. First, I thank my supervisor, Dr. Tony White, who helped at each step of the process to push me to make difficult decisions, to approach problems scientifically and rigourously, and to learn as much as I could absorb. He has helped me to become a better scientist.

I thank my parents, Carole and Stanley, for their endless support and love. I could not have finished without their moral support, unwaivering faith in me, and efforts to inspire me to be organized and efficient. I owe them everything.

I thank my love, Natalie, for her love and support, and constant understanding and patience regarding my long hours in relative seclusion.

I thank my Grandmother, Wanda, for her support and pride in me, and for her continuous interest in my work.

I thank Rick R. for his support and advice, which helped me to work through my thesis while not losing sight of the path beyond.

I would also like to thank several professors for offering their impressions of my thesis early on in the process, including: Professor Manuel Baez, of Carleton University’s Department of Architecture, Professor John Lewis of the University of Waterloo’s School of Planning, and Nick Moylan of Algonquin College regarding GIS.

Finally, I would like to thank Adam M., Matthew C., Andrew F., and Jeremy K. for being great friends during this process by enduring many phone conversations, in which they listened to me air new ideas or complain about technical obstacles, and for putting up with my relative seclusion.
Chapter 1

Introduction

1.1 Overview

Cities are complex entities which play important roles in the lives of the many people who inhabit them. The process of creating city designs is a complex, time-consuming endeavour, pursued by several different groups. Though procedural techniques have been developed to speed up this process, virtually none enable the creation of designs based on multiple existing designs. This thesis presents CityBreeder, a system which enables the rapid, user-guided development of city designs based on the blending of multiple existing city designs. Almost no previous research has been conducted regarding this capability.

This capability is achieved through the use of Evolutionary Computation, which is used to discover the genetic representation of existing city designs derived from real city data obtained from OpenStreetMap. Once discovered, these cities can be ‘bred’ together, creating new offspring designs. More of this thesis is concerned with the first portion of this task: the discovery of the genetic representations of real city designs. The combination of these cities is given less attention, but is explored through several demonstrations which show this capability is achieved.

More specifically, CityBreeder employs Genetic Programming on a layered quadtree genotype representation to create phenotype city designs consisting of road layouts comprised of nodes and edges. Additionally, a genotype-to-phenotype expression mechanism, genetic operators and a fitness function employing computational geometry techniques are presented and tested, all of which are tailored to the city design context. Experiments and examples are shown which analyze the system’s representation and operators using simple, artificially constructed data, as well as through experiments showing the system functioning with data derived from real cities.

In this chapter, the problem statement is articulated and discussed, and the organization of the rest of the thesis is presented.
1.2 The Problem Statement

The problem statement for this thesis is: how can one create city designs that are a blend of existing designs which are derived from real cities?

A solution to this problem will therefore: enable the rapid, user-guided development of city designs based on the blending of multiple existing designs derived from real cities.

References to the problem statement are made throughout this thesis, and tend to refer to the latter ‘solution’ form.

1.2.1 Motivation

Previously Unanswered

Previous work related to computer-based city design has produced many different tools and techniques to make city design more efficient. Particularly, many procedural techniques have been developed to reduce the amount of time needed to create new designs. However, as will be shown in the Related Work (Chapter 3), no previous approach has offered a complete answer to this problem.

Worth Answering

Cities are important to the lives of much of the world’s population, and their design affects the lives of their many inhabitants. The process of designing cities and neighbourhoods typically involves a large amount of time-intensive, manual work. City designers wishing to combine features of existing cities, whether explicitly or subtly may have difficulty doing so, as the amount of work in identifying traits and methods of blending them manually would be time-consuming and inconsistent.

It could be desirable, particularly in multi-cultural countries, to design a neighbourhood that mixed features of typical existing neighbourhoods in that country with those from the neighbourhood’s ethnic community’s homeland. For instance, a designer could redesign a city’s Little Italy neighbourhood by blending designs derived from that city and Rome.

Alternatively, blending designs from cities in the same country would still benefit more conventional city design, allowing a designer to rapidly generate new designs without the
need for specialized training, as they can essentially ‘point-and-click’ on the designs they like at each generation, breeding them until a desired solution is found.

While breeding cities is of particular value to certain groups of users, the preceding step of discovering the genes for a city design presents additional scientific interest and value. Along with an examination of the representation and genetic operators, this work should provide interest to researchers in both the evolutionary computation and city design fields.

Enabling this novel approach to city design, and the scientific value of exploring a ‘genetic’ understanding of city design, is why the problem is worth answering.

1.2.2 Solving the Problem

This thesis solves the problem by first demonstrating that the system’s design and implementation meets all of the requirements embedded in the problem statement, thereby solving it in principle. This is followed by experiments and examples which demonstrate that the system answers the problem statement well.

In the first portion, CityBreeder’s pipeline (its overall process containing multiple elements where the output of one acts as the input to another) is articulated, which consists of two phases: Discovering City Genes, and Breeding Cities. This is followed by an examination of the main components used in each phase, consisting of a mechanism of importing data from OpenStreetMap, and evolutionary components including genetic operators, an expression mechanism and fitness function.

The second portion explores how well the system performs through experiments which examine the fitness and realism achieved through evolution in Discovering City Genes, operating both with real city data, and with simple contrived data designed to test specific qualities of the system. Additional examples illustrate properties of the expression mechanism, examining how expressive and robust it is. Finally, demonstrations and analysis are presented which show the system creating new city designs by breeding designs based on real city data.

1.3 Scope

City designs may contain many features, and an investigation of the problem statement might be approached in different ways. In this section, the scope of the problem, and specific
requirements for a solution are outlined.

1.3.1 Features

There are many features that can be modelled for city designs, depending on the application. These features are discussed in detail in 2.2.2. This thesis examines the road network only, and treats such a network as constituting a city design. The road network can be represented in many ways, and the approach taken in this thesis is discussed in 4.3.2.

Features which might affect road networks such as water boundaries, and boundaries caused by other natural phenomena are not modelled in the representation, but are shown in one of the experimentation chapters (Chapter 7) manually overlaid on some of the evolved designs. How these and other features might be incorporated into the representation is discussed in Future Work (9.2).

1.3.2 System Design and Implementation

As previously mentioned, this thesis offers a system design and implementation as a part of the answer to the problem statement. Though the system design and implementation are discussed, the software itself is not. Rather, the focus is on the system’s architecture and algorithms. Consequently, the implementation is not considered from a software engineering perspective.

1.3.3 Experimentation

In order to determine how well the software performs, and consequently how well it answers the problem statement, experiments were run, and are presented and discussed. A number of domain-specific operators are presented, analyzed, and used in experimentation. However, an exhaustive exploration of the parameter space, achieved by running experiments with many different values for these operators and other evolutionary settings, would require an excessively large amount of time, and is not performed. Instead, experiments were run using parameter values which are considered ‘reasonable’ within the genetic programming literature. These choices concerning parameter values are discussed in 6.2.

It should also be noted that the system presented in this thesis is capable of working with city designs derived from ‘entire’ cities. However, in the experiments conducted, city designs
are used which are derived from smaller segments of cities, including portions of multiple neighbourhoods.

Additionally, it should be noted that the evolutionary experiments conducted focus on the macro, population level. Individual lineages and interactions are not explored in experimentation.

1.4 Definitions

The terms used in the problem statement are defined below so that the solution offered in this thesis along with approaches in the related works can be evaluated against satisfying the problem statement.

1.4.1 Rapid

The motivation for having a rapid system is to enable users to create city designs quickly without having to spend much time creating the actual result or having to prepare much input data for the system. Therefore, this requirement can be decomposed into 2 components:

- **Fast**: the system can create a city design in under one minute, and the user can create a ‘final’ design in under one hour.

- **Minimal Input Data Effort**: Some input data is permissible, but it must be in a form that requires minimal time or energy to acquire or produce. If it requires expert training to produce or is not freely available, it does not qualify.

1.4.2 User-Guided

The process of city design is creative, and the user should not be cut out of the process. This requirement creates a tension with the previous one—of having the system be rapid. A balance is therefore desirable which allows the user to guide the process in a way that does not slow down the system beyond the thresholds previously established.

For the process to be user-guided, it must incorporate user feedback within the system. Specifically, producing a result and forcing the user to modify input data and run the system again does not qualify. Furthermore, the blending process itself must be user-guided.
1.4.3 City Designs

As previously mentioned, there are many possible features which could be included in city designs, but in this thesis, only road networks are considered.

Furthermore, road networks are treated as consisting of vector or raster data where the roads are clearly separable from other design elements. Aerial photography (without additional accompanying road data) is not considered as constituting a road network.

1.4.4 Blending of Multiple Existing City Designs

For a solution to be a blend of multiple existing city designs, it should combine features of those existing designs. However, the blending should not merely consist of discrete elements that have been cut and pasted together from different elements (a “mixture”), but instead should be a deeper blending of the properties of those elements, creating new design elements.

There are few enough previous works which employ blending that their specific approaches are examined in detail in Chapter 3, where this distinction is further discussed.

1.4.5 Derived from Real Cities

The multiple existing city designs which are blended should be derived in some way from real city data.

1.5 Contributions

The main scientific contributions offered by this thesis are:

- A genetic representation tailored to the city design context, consisting of genotype, phenotype and expression mechanism
- A fitness function for the city design context, based on the comparison of categorical maps created from the properties of enclosures within a city design’s road network
- A demonstration of the creation of new city designs based on existing designs through the use of evolutionary computation
- The discovery of the city genes for several designs based on real cities, and on simple exemplars, with an analysis of the results
• A systematic evaluation of the representation, focusing on expression, operators, and fitness function

1.6 Thesis Organization

The remainder of this thesis is organized as follows below. However, one note to the reader: the page count on this document is rather high, but please keep in mind that the second half has far more images and is digested much more quickly.

Background Information

In Chapter 2, additional background information is offered which may be helpful in understanding the rest of the thesis. Topics discussed include: city design, evolutionary computation, geographic information systems (GIS), computational geometry, categorical maps, and three additional techniques which are referenced in the Related Work.

Related Work

In Chapter 3, a review of the state of the art is presented for computer-based city design. Procedural approaches, including those which blend existing designs and those that do not, and evolutionary approaches are both discussed. This is followed by a brief discussion concerning other city features which are modelled in related approaches.

System Design and Implementation

In Chapter 4, CityBreeder’s design and implementation is presented and discussed. Its pipeline is illustrated, consisting of its main two phases: Discovering City Genes and Breeding Cities, and also its Import mechanism. Next, the genetic components of the system are presented, including: genotype, phenotype, expression mechanism, genetic operators and fitness function. The chapter ends with a discussion of how the system’s design and implementation answers the problem statement in principle.

Experiments and Examples

Chapters 5 through 8 constitute the experimentation and examples chapters. In Chapter 5, Expression Examples, examples are presented that explore how expressive the genetic
representation is, through examples which illustrate and analyze the expression of a number of genotypes into phenotypes. The examples are selected to show how different scale and angle layer ‘surfaces,’ along with changes to the angle continuity window parameter, affect the resulting phenotype. The examples presented fall into the categories: Simple Growth, Complex Growth, Simple Rotation, and Complex Rotation and Angle Contiguity Values.

Before deploying the system to discover the genes of real cities, experiments are presented in Chapter 6 (Simple Evolution), which demonstrate how effectively CityBreeder is able to discover the genes of simple, contrived examples. It is shown that it performs well, providing some evidence of its effectiveness and utility. Additionally, experiments are performed on a simple target, using each operator in isolation. While this does not constitute an extensive mapping of the parameter space, it does offer some insight into the utility of each operator.

Before existing real city designs can be bred together to produce new ones, the genes of those existing designs must be discovered. In Chapter 7 (Discovering City Genes), experiments are presented which endeavour to discover the genes of segments, or neighbourhoods, from 3 real cities: Ottawa, New York, Paris. It is shown to find good results with Ottawa and New York, but has more difficulty with Paris. The reasons for this are discussed.

With the genetic material discovered in Discovery of City Genes on hand, Chapter 8 (Breeding Cities) provides demonstrations of the city breeding process, showing how the system works, and illustrating how the user can direct the evolution in various directions to evolve a particular design. These demonstrations illustrate that specific traits can be selected for and propagated over a number of generations, and that characteristics of parent cities can be seen in their offspring. Additionally, several detailed examples are presented, showing the recombination of genetic material from two real city designs.

Conclusion

Chapter 9 concludes the thesis with a reiteration of the scientific contributions made, revisits how the problem statement is answered, and explores a variety of potential future work including improvements, additional experimentation, and extensions within and beyond the city design context.
Chapter 2

Background Information

2.1 Introduction

Although this thesis assumes its reader is proficient with computer science, there are several additional and specialized topics that are of importance to this thesis, which the reader may not be familiar with. These topics are discussed in this chapter, and include: city design, evolutionary computation, geographic information systems (GIS), computational geometry, categorical maps, and three additional techniques which are referenced in the Related Work.

2.2 City Design

Cities are complex entities whose designs are created by several different groups of people who model different city features depending on their purpose. In this section, the particular view of city design relevant to the problem statement is explored and situated within the larger field.

2.2.1 Terminology

In the computer-based procedural city design literature, which is discussed in the next chapter, a variety of terms appear which refer to city design including: city modelling, modelling urban environments, and creating urban layouts. Some of these terms have been employed to refer to designs that include certain features. For example, Interactive Example-Based Urban Layout Synthesis uses the term ‘structured urban layout data’ to refer to vector street data paired with aerial imagery [5].

In this thesis, any approach which captures a city feature and produces a design is considered city design.
2.2.2 Features

As mentioned in Chapter 1, this thesis is concerned with producing city designs consisting of road networks. Most approaches in the Related Work include road networks in their designs. However, it is useful to consider what other features of a city might appear in a design, as some of these appear in Chapter 3, and others are discussed further in the Future Work.

Some of the many city features that have been represented in designs include: road networks, other transportation networks, zoning, building footprints, 3d building geometry or architecture, vegetation, and terrain. Different groups will prefer some of these features more than others. Specific groups and features that interest them are discussed later in this chapter in 2.2.4.

2.2.3 Data Types

As the problem statement requires that the design process incorporate data derived from real cities, it is necessary to consider different types of such data. Many types of city data exist, some of which are:

- **GIS data** - a broad characterization for geographic data. It is used in this thesis, and is discussed in detail.

- **Aerial imagery** - top-down images, such as those as seen in Google Maps’ satellite imagery.

- **Ground-level imagery** - photographs taken at ground level. This data can used in complex ways, such as with Google Maps’ Street View.

- **Real-time data** - data describing elements of a city in real time. This may reflect many possible features, one example being traffic information.

GIS data comprises a variety of types of data such as road networks, building footprints, natural boundaries, and statistical information. GIS data is used in this thesis, and is discussed later in this chapter in 2.4.

Different forms of imagery are useful in constructing 3d city designs, and real-time data can be useful in creating or modifying designs to reflect how a city space is used. Imagery and real-time data are not used in this thesis, and are not explored further.
2.2.4 Interested Groups

The creation of city designs is of interest to a number of different groups including: *urban designers*, *urban planners*, *visual effects artists*, and *game level designers*. It is worth briefly examining these groups as their relative potential interest in this work helps to explain its utility.

Urban planners and designers are likely to be the groups most interested in the work presented in this thesis for reasons that are explained below. Visual effects artists and game level designers may be interested, but would likely be more interested in the potential future work and extensions of *CityBreeder* which are discussed in the Future Work section (9.2).

**Urban Designers**

Urban designers and planners are two closely related groups whose designs are based on, and lead to the creation of, real cities. Urban design has been defined as follows:

“Urban design is city-building. It brings together the many different parts and pieces of an environment to create a place. At its core is design, an inventive process that draws upon the techniques of many different disciplines to create beautiful, felicitous environments. Therefore, urban designers must be generalists capable of bringing together diverse specialists and technicians to create unified vision.” [32, p. 17]

It has also been characterized as:

“Urban design is an integral part of the process of city and regional planning. It is primarily and essentially three-dimensional design but must also deal with the non-visual aspects of environment such as noise, smell or feelings of danger and safety, which contribute significantly to the character of an area. Its major characteristic is the arrangement of the physical objects and human activities which make up the environment; this space and the relationships of elements in it is essentially external, as distinct from internal space. Urban design includes a concern for the relationship of new development to existing city form as much as to the social, political and economic demands and resources available. It is
equally concerned with the relationship of different forms of movement to urban development.” [34, p. 7]

Urban design is similar to urban planning. Both groups’ potential interest in this work is summarized at the end of that section.

**Urban Planners**

Urban planning has been defined as follows:

“Modern urban planning is a mixture of science and art. It encompasses many different disciplines and brings them all under a single umbrella. The simplest definition of urban planning is that it is the organisation of all elements of a city or other urban environments. When one thinks about the various elements that make up a city, then the complexity of urban planning seems transparent...

Urban planning is thus a wide and multidimensional concept, embracing multi-disciplinary interests ranging through land-use to transportation, environment to social and economic life, and from neighbourhood level to regional or national level. It, in effect, considers everything involving urban design and macro, or national, planning. This includes policy issues involving local institutions up to central government, as well as public-private partnerships, community involvement in planning processes and urban governance.” [45, p. xiii]

It is a broad, interdisciplinary field, and as a result has a somewhat loosely defined scope:

“Urban planning is a relatively young academic discipline and, despite its storiied genes, lacks an extensive, established canon on which to rest its laurels. Its youth affords it the flexibility to take on varied guises: an upstart social science; a boundary-spanning source of professional knowledge; and a fraternity of generalists, problem-solvers, and idealists, many being migrants from other, more traditional disciplines.” [86]

The number of subdomains within urban planning is not precisely defined, but one source lists them (non-exhaustively) as the planning of: transportation, air quality, water quality,
solid waste, health services, recreation facilities; as well as: site planning, project planning, comprehensive planning, and master planning [57].

With the tools traditionally associated with urban planning and design, the creation of city designs is a slow process, requiring a great deal of expertise and effort. A number of procedural techniques have been developed to speed up this process, which are discussed in Chapter 3. However, few approaches offer the ability to create new city designs by blending existing ones. The work offered in this thesis would be of interest because it presents such an approach, which can be used quickly, and potentially without as much training and expertise as required by other techniques. All of these advantages would save urban planners and designers time and money.

**Visual Effects Artists**

In many films, urban settings play an important role. Visual effects artists may be interested in this work, but would likely be particularly interested in its potential extensions, which are discussed in the Future Work (9.2). There are often times when it is more practical to use visual effects (also known as VFX or Visual F/X) to simulate the urban environment or parts thereof, rather than to use real ones. *The VES Handbook of Visual Effects: Industry Standard VFX Practices and Procedures* lists 3 instances where it is preferable to use visual effects [58, p. 2-3]:

- “When there is absolutely no practical way to film the scenes described by the script or required by the actor.”
- “When you could do the scene practically, but doing so might place someone’s life at risk.”
- “When it is more cost effective or practical to utilize a visual effect than to film a scene for real, due to issues of scale or location (or both).”

Historically, the use of models and matte paintings were employed when studios needed video of cities or segments thereof. With the rise of computing power, 3d modelling, rendering and animation software, such as *Autodesk 3ds Max* [10], *Autodesk Maya* [11], and *Blender* [28] have become common tools used to visually emulate urban environments. It is cheaper and safer to move city blocks or demolish buildings when it is done on the computer in 3d.
While certain movies may simply require recreating a portion of a city with physical or 3d models, there are also times when models of entirely new cities are needed. For instance, science fiction movies often feature fantastical cities, whose designs are not based on existing designs. Both the old fashion modelling and many of the 3d modelling tools still require a great deal of artistic talent and hours of laborious work to achieve desired results. A tool that allowed for faster creation of city models would save film studios significant time and money.

City designs consisting of road networks, as presented in this thesis, could be of use in planning city layouts to later be filled with physical or 3d models. If the approach taken in this thesis were extended to enable the creation of 3d city designs, it would be of great interest to this group. This extension is discussed further in Future Work in 9.2.

Game Level Designers

Cities and urban contexts form the environments of many video games. While some games such as SimCity [8] or CitiesXL [40] focus their attention precisely on the city itself, many others incorporate it into their background fabric. Whether it is an early 2d side-scrolling game where a cityscape slides along in the background, a flight simulator where the user flies above a simplified model of a real city, or a modern 3d first-person-shooter where the user navigates a full, photorealistic 3d city on foot and can enter buildings, urban environments are found in almost any game. Level designers create these settings. They “create the spaces and environments that you move through and experience as you play video games” [14, p. x].

One question game designers must initially answer is to what level of abstraction they can model an urban context. Not all games require 3d buildings a player can enter, with a range of natural phenomena that are simulated. Different tools are appropriate in different contexts. 3d modelling tools may be used to design portions of the environment, and are often used in conjunction with one of many game-specific editors, such as the Valve Hammer Editor [78] in the Source SDK [77], CryENGINE [15], and the Unreal Development Kit [30]. While many of these tools simplify the process of creating game levels, they often still require technical and artistic training. Simple, procedural tools that can simplify the process of creating large-scale city designs would save game studios time and money. It would also allow amateur game designers to create levels quickly, without requiring a high level of artistic
and technical computer knowledge.

Consequently, the work presented in this thesis, and particularly its potential future work, may be of interest to game level designers. City designs consisting of road networks may be relevant to map creation for table-top role playing games. Also, as described for VFX artists, the current implementation could serve these users by allowing them to create the initial plan upon which to model further in 3d. Similarly, if the system is extended to include 3d features, as discussed in Future Work, it would be of great interest to level designers.

2.3 Evolutionary Computation

Evolutionary Computation (EC) is a subdomain within computer science, which is based on the mechanisms of evolution. It involves expressing potential solutions to a problem ‘genetically,’ consisting of genotype and phenotype representations, a mechanism for expressing a given genotype into a phenotype, and a method of evaluating a phenotype’s fitness. Using a population of individuals, the fitness of each member is determined, and the fittest are reproduced more frequently. Together with genetic operators such as mutation and crossover, this produces fitter individuals over a number of generations. Put another way, it contains “a population of individuals that reproduce with inheritance... genetic variation [and] natural selection.” [18, p. 87]

Individuals unfamiliar with these evolutionary components may better understand them by considering their biological analogues. An individual human (phenotype) is created from their DNA or genes (genotype) through a complex expression and development process. Once mature, they may sexually reproduce (whether this occurs may be influenced by some measure of fitness), which creates new offspring through genetic recombination.

This overall pipeline shown in Algorithm 1, and is adapted from [63, p. 5]. This approached is used in the thesis, and is specifically discussed in 4.2.

2.3.1 Genetic Programming

There are several forms of evolutionary computation which follow this approach. The approach used in this thesis is Genetic Programming (GP), which was originally proposed by Nichael Cramer, and then subsequently developed and formalized by John Koza [18, p. 105].
Algorithm 1 Evolutionary Computation Pipeline

1: initialize population with random candidate solutions
2: evaluate fitness of each individual
3: while termination condition not satisfied do
4:   select parents
5:   recombine pairs of parents
6:   mutate resulting offspring
7:   evaluate fitness of offspring
8:   select individuals for next generation
9: end while

GP is an extension of Genetic Algorithms (GAs). While GAs tend to work on a genotype representation consisting of fixed-length bit-strings, GP genotypes tend to be tree-like structures (or tree-like S-expressions), evolving actual functions or computer programs. As a result, mutation and crossover operators have been developed that work on trees structures.

2.3.2 Operators

The following are the main classes of operators and mechanisms used in GP:

- **Initialization** - creates the initial population
- **Reproduction** - copies an individual exactly
- **Crossover** - recombines the genetic material from multiple parents
- **Mutation** - alters elements within an individual’s genotype
- **Selection** - chooses individuals from a population (used at two points in the evolutionary process)

Initialization, and selection operate at the level of individuals within a population, while reproduction, crossover, and mutation are all genetic operators which operate ‘within’ individuals.
Initialization

Initialization techniques determine how the initial population is created. They may be created randomly to maximize diversity, or the initial population may be created using known information, adding bias which may place initial solutions closer to an expected ideal. In this thesis, two initialization techniques are used: Ramped Half-and-Half Initialization, which is presented below, and Prepopulated Initialization, which is discussed in Chapter 4.

Ramped Half-and-Half Initialization

Ramped Half-and-Half Initialization is a standard initialization technique which initializes half of its individuals according to $\text{Grow}$, which produces a random tree, and half according to $\text{Full}$, which produces a tree with the maximum number of nodes. Each of these operates up to a specified maximum depth, with Full having leaves only at the maximum depth. Ramped Half-and-Half Initialization creates a diverse population using each of these techniques with varying levels of depth.

It should be noted that there is another process called ‘Grow’ used in this thesis, which is part of the expression mechanism and is introduced in Chapter 4. To avoid confusion, Ramped Half-and-Half Initialization’s Grow function is referred to as $\text{Grow-Initialization}$ in algorithms.

The algorithms for these three mechanisms appear below, with Ramped Half-and-Half Initialization shown in Algorithm 2, and Full and Grow in Algorithms 3 and 4 respectively. These algorithms are adapted from Essentials of Metaheuristics [50, p. 72-73]. In Grow and Full, $\text{functionSet}$ and $\text{leafSet}$ represent lists of the non-leaf and leaf node types used by a given representation.

Reproduction

Koza has suggested that the two primary genetic operations in GP are reproduction and crossover [44, p. 99]. While experiments are run with high crossover probabilities, reproduction occurs at those times when crossover does not operate, copying the parents as children. This is not modelled explicitly in Chapter 4, as it is considered to be a part of crossover.
Algorithm 2 Ramped-Half-and-Half Initialization

1: function RAMPED-HALF-AND-HALF(rampDepths, numIndividualsPerRamp, functionSet, leafSet)
2:     population ← empty list
3: for all depths in rampDepths do
4:     loop numIndividualsPerRamp times
5:         if depth < a random value number ∈ [0.0..1.0] then
6:             add GROW-INITIALIZATION(1, depth, functionSet, leafSet) to population
7:         else
8:             add FULL(1, depth, functionSet, leafSet) to population
9:     end if
10: end loop
11: end for
12: return population
13: end function

Algorithm 3 Full Initialization

1: function FULL(depth, maxDepth, functionSet, leafSet)
2:     if depth ≥ maxDepth then
3:         return an instance of a randomly-chosen leaf node from leafSet
4:     else
5:         n ← an instance of a randomly-chosen non-leaf node from functionSet
6:         l ← number of child nodes expected for n
7: for i ← 1 → l do
8:     individual ← FULL(depth + 1, maxDepth, functionSet, leafSet)
9:     add individual to n as a child node
10: end for
11: return n
12: end if
13: end function
Algorithm 4 Grow Initialization

1: function Grow-Initialization(depth, maxDepth, functionSet, leafSet)
2:  
3:    if depth ≥ maxDepth then
4:        return an instance of a randomly-chosen leaf node from leafSet
5:    else
6:        n ← an instance of a randomly-chosen node from functionSet ∪ leafSet
7:        l ← number of child nodes expected for n
8:        for i ← 1 → l do
9:            individual ← Grow(depth + 1, maxDepth, functionSet, leafSet)
10:               add individual to n as a child node
11:         end for
12:     return n
13: end function

Crossover

Crossover is a genetic operator used to recombine the genetic material from multiple (typically 2) parents. Crossover creates one or more children by copying portions from the genotypes of both parents into the new individuals’ genotypes. How the portions are selected varies between different forms of crossover. The two forms of crossover used in this thesis are: Basic Crossover and Weak Context-Preserving Crossover.

Basic Crossover

A basic form of GP crossover simply involves choosing any node (and by extension its whole subtree) from each parent and swapping them, producing two children [62, p. 36]. The algorithm is presented in Algorithm 5.

Weak Context-Preserving Crossover

Weak Context-Preserving Crossover operates in a similar fashion to Basic Crossover but attempts to retain additional spatial information by selecting swap points in similar regions of the genotype. This technique makes use of the notion of a node-coordinate, which is the
Algorithm 5 Basic Crossover

1: function Basic-Crossover(parentA, parentB)
2:    childA ← copy of parentA
3:    childB ← copy of parentB
4:    subtreeRootA ← randomly selected node in childA
5:    subtreeRootB ← randomly selected node in childB
6:    swap subtreeRootA and subtreeRootB
7:    return childA and childB
8: end function

“same path from root to node” [62, p. 41]. With Weak Context-Preserving Crossover, “only subtrees with common node-coordinate[s] can have [crossover] points but: any subtree of corresponding node coordinate subtree can be chosen for swap” [62, p. 41]. The difference then is that Basic Crossover introduces a greater degree of randomness, or global exploration into the search, while Weak Context-Preserving Crossover ‘retains’ more spatial information.

One last distinction worth mentioning is why the mechanism is referred to as ‘Weak’ Context Preserving Crossover. The reason is that there also exists a Strong Context Preserving Crossover, which requires that the swap points must have the exact same node-coordinates (at the same depth; they cannot merely be along the same path as with the ‘Weak’ variant). As a result, Weak Context-Preserving Crossover is more flexible, and is more powerful in its ability to introduce changes.

An example may help to illustrate these differences more clearly. Consider the two parents shown in Figure 2.1. Given a swap point found in Parent A (highlighted in red), Figure 2.2 shows which nodes are valid potential swap points with each of these three operators.
Figure 2.1: Crossover Illustration Parents
One possible specification of Weak Context-Preserving Crossover appears in Algorithm 6. This is the manner in which the operator was implemented in the thesis. It retains common node-coordinates between two swap points by beginning at the root in each individual and proceeding through each tree making identical ‘turns’ while searching for each swap point. This approach requires one parameter which specifies the probability of stopping and selecting a swap point versus traversing the tree deeper. This is defined as:

\[ \text{Stop at Terminal Probability} \equiv \kappa := 0.5 \]
Algorithm 6 Weak Context-Preserving Crossover

1: function Weak-Context-Preserving-Crossover($parentA$, $parentB$)
2: \hspace{1em} $childA \leftarrow$ copy of $parentA$
3: \hspace{1em} $childB \leftarrow$ copy of $parentB$
4: \hspace{1em} $searchNodeA \leftarrow$ $childA$’s root
5: \hspace{1em} $searchNodeB \leftarrow$ $childB$’s root
6: \hspace{1em} $subtreeRootA \leftarrow$ $subtreeRootB \leftarrow$ null
7: \hspace{1em} while $subtreeRootA$ and $subtreeRootB$ are not both assigned nodes do
8: \hspace{2em} $childNum \leftarrow$ random number $\in [0..(# \text{ of } searchNodeA \text{’s children})]$ \hspace{1em} 
9: \hspace{2em} for $i \leftarrow 0 \rightarrow 1$ do \hspace{1em} 
10: \hspace{3em} if $i = 0$ then \hspace{1em} 
11: \hspace{4em} $subtreeRoot \leftarrow$ $subtreeRootA$
12: \hspace{4em} $searchNode \leftarrow$ $searchNodeA$
13: \hspace{3em} else \hspace{1em} 
14: \hspace{4em} $subtreeRoot \leftarrow$ $subtreeRootB$
15: \hspace{4em} $searchNode \leftarrow$ $searchNodeB$
16: \hspace{2em} end if \hspace{1em} 
17: \hspace{2em} if $subtreeRoot$ has not been assigned a node then \hspace{1em} 
18: \hspace{3em} if $searchNode$ is a terminal node then \hspace{1em} 
19: \hspace{4em} $subtreeRoot \leftarrow$ $searchNode$
20: \hspace{3em} else if a random value $\in [0.0..1.0] < \kappa$ then \hspace{1em} 
21: \hspace{4em} $subtreeRoot \leftarrow$ $searchNode$
22: \hspace{3em} else \hspace{1em} 
23: \hspace{4em} $searchNode \leftarrow$ $searchNode$’s child in position $childNum$
24: \hspace{3em} end if \hspace{1em} 
25: \hspace{3em} end if \hspace{1em} 
26: \hspace{2em} end for \hspace{1em} 
27: \hspace{1em} end while \hspace{1em} 
28: \hspace{1em} swap $subtreeRootA$ and $subtreeRootB$
29: \hspace{1em} return $childA$ and $childB$
30: end function
Mutation

Mutation is a genetic operator that takes an individual and modifies its genotype. Mutations can either be harmful or beneficial, and the effect is only known once the mutated individual’s new fitness is determined. This operation allows evolution to explore new possible forms that would not be attainable through crossover alone. Though mutation has been characterized as less important in GP than crossover [44, p. 105], without mutation, it is not possible to find a leaf value which is not already present in the initial population. Additionally, crossover can lead to important genetic traits disappearing from one generation to the next, some of which can only be rediscovered through mutation.

As with crossover, many different mutation operators exist. In this thesis, 5 mutation operators are used and are presented and discussed in detail in Chapter 4. However, several of these are based on the canonical mutation operators of Permutation Mutation and Gaussian Value Mutation.

Permutation Mutation

Permutation Mutation is a form of mutation designed for tree-like genotype representations, operating on non-leaf nodes. Generally speaking, Permutation Mutation takes the leaves of a subtree, and randomly rearranges their ordering. This would only have an effect on representations in which the order of subtrees matters, which tends to be the case in much of the GP literature. The two operators used that are based on Permutation Mutation are discussed in 4.4.2. An illustration of this operator is shown in Figure 2.3.

Figure 2.3: Permutation Mutation Illustration
Gaussian Value Mutation

Gaussian Value Mutation is a common form of mutation which operates on the leaf nodes of tree-like genetic representations, changing their values by a certain amount according to the Gaussian function, which returns values based on a specified mean and standard deviation. An illustration of this operator is shown in Figure 2.4.

![Figure 2.4: Gaussian Mutation Illustration](image)

Selection

Selection occurs at two points in the evolutionary process. From the pipeline shown in Algorithm 1, these two correspond to select parents and select individuals for next generation. These two rounds of selection may also be characterized as: “who gets to mate?” and “who gets to survive?” [63, p. 9]

A large number of selection mechanisms have been developed. Two mechanisms which are used in this thesis are Rank Selection and Uniform Selection.

Fitness Function

For evolution to improve the fitness of a population over time, it is necessary to select individuals in at least one of the two rounds of selection in a manner proportional to their fitness. Doing so will see individuals with higher fitness reproducing more, which tends to see their beneficial traits propagated across generations.

Fitness may be evaluated in a number of ways, either as an explicit mathematical function, through simulation which tests an individual, or through subjective assessment. However, the overall approach remains the same: given individual $a$, return fitness $f(a)$. 
Rank Selection

Rank Selection ranks the individuals in a population based on their fitness before selecting from them in a manner proportional to their rank. Rank Selection may be used to select as many individuals as desired after the preparatory ranking process is complete, which both ranks them and generates an array of cumulative distribution probability values. The algorithms for Rank Selection and its preparatory process are shown in Algorithms 7 and 8 respectively.

Algorithm 7 Rank Selection

1: function Rank-Selection(rankedPopulation, cumulativeProbabilities)
2:     randomValue ← random value ∈ [0.0..1.0]
3:     for all probability values in cumulativeProbabilities do
4:         if probability is the first value in cumulativeProbabilities and randomValue ≤ probability then
5:             return the corresponding member in rankedPopulation
6:         else if the previous probability value < randomValue < probability then
7:             return the corresponding member in rankedPopulation
8:         end if
9:     end for
10: end function

Uniform Selection

Uniform selection selects an individual from the population uniformly: randomly with equal probability. The algorithm is shown in Algorithm 9.

2.4 Geographic Information Systems

Geographic Information Systems (GIS) is “an information system that is designed to work with data referenced by spatial or geographic coordinates. In other words, a GIS is both a database system with specific capabilities for spatially-referenced data, as well as a set of operations for working with the data” [70].
Algorithm 8 Preparation for Rank Selection

1: function PREPARATION-FOR-RANK-SELECTION(unrankedPopulation)
2:     rankedPopulation ← sort unrankedPopulation by descending fitness
3:     totalFitness ← Σ rankedPopulation’s fitness values
4:     for i ← 0 → population’s size − 1 do
5:         cumulativeProbabilities’s i value ← i / totalFitness
6:     if i ≠ 0 then
7:         cumulativeProbabilities’s i value ← cumulativeProbabilities’s i value + cumulativeProbability’s i − 1 value
8:     end if
9:     end for
10:    return rankedPopulation and cumulativeProbabilities
11: end function

Algorithm 9 Uniform Selection

1: function UNIFORM-SELECTION(population)
2:     individual ← random member of population
3:     return individual
4: end function
GIS is used by a number of fields. The most obvious field might be geography, but it is also closely related to urban planning and design. *Urban Planning and Development Applications of GIS* discusses GIS methodologies and applications in Urban Planning, including “GIS technology and implementation, remote sensing, trends in spatial databases... implementation of linear referencing systems in GIS... regional planning, transportation, public utilities, stormwater and waste management, cultural and resources management, environmental assessment, socioeconomic development, and academic education” [20, p. iii].

The leading GIS software company, *Esri* [24], provides a range of software such as *ArcGIS* [21] which is the industry standard. Other software may be used, including free software such as *ArcGIS Explorer Desktop* [22], *Quantum GIS* [76], or *GRASS GIS* [75], among others. Additionally, free tools exist to help developers who wish to manipulate GIS data, such as *GeoTools* [31].

GIS is particularly relevant in this thesis as the problem statement requires that existing designs which are derived from real city data be blended. The data for these existing designs is gathered from GIS data sources.

2.4.1 Data Sources

Many sources of GIS data are available online. The average consumer of this type of information will likely use a source such as *Google Maps* [33], *Bing Maps* [53], or *Apple’s Maps* [7]. However, these sources make it difficult or impossible to extract information, a capability needed to create city designs from these sources.

Data is offered by organizations and governments at various levels of organization. For example, this type of data is available for Canada from at least: 4 sources at the federal level, 4 at the provincial level, and 30 at the municipal level. These sources are listed in Appendix A. Similar data for other countries can be found from several sources including *Datacatalogues.org* [16], *GeoComm* [38], and *Wikipedia’s Open Data page* [90].

Given the large number of data sources for Canada alone, using these ‘non-worldwide’ data sources makes it difficult to quickly and consistently extract data for multiple cities around the world. Furthermore, there are other drawbacks to assembling data from a variety of sources in this manner. It is difficult to unify the data offered into a single coherent source from which individual cities can be extracted, which one might desire as the speed
and ease of data procurement are elements of the problem statement. Also, data concerning different features are offered from different repositories with different degrees of accuracy and resolution. Additionally, while some of these data sources contain recent data, others contain data which may be years or decades old, and no longer accurate. To create a system that can breed multiple city designs, it is advantageous to have data which is consistent. As evidenced by the large number of data sources for Canada, it would be labour-intensive and time-consuming to assemble data from multiple sources, and may yield city designs with varying levels of quality.

2.4.2 OpenStreetMap

In this thesis, existing city designs are created from data extracted from OpenStreetMap [61], which provides free mapping data for the world. OpenStreetMap, often abbreviated OSM, is a crowd-sourced mapping service. With a large number of users, the data can be up-to-date, particularly in cities which have larger populations than rural areas. OpenStreetMap also allows users to extract data openly from its service provided it is used in conjunction with its Open Data Commons Open Database License [87].

OpenStreetMap limits its exports based on the number of nodes contained within a specified bounding box. To circumvent this limitation users may also download the offered planetary data file Planet.osm [89], from which larger bounding boxes can be exported using additional software such as Osmosis [88]. In this thesis, when data is referenced as having been exported from OpenStreetMap, it should be interpreted as having been exported either from the OpenStreetMap website directly, or through Osmosis.

OpenStreetMap data reflects a range of features, such as road networks, natural boundaries, and with less complete coverage: building footprints, height data, addresses, and traffic lights. Of interest to this thesis is its road network information, which appears to be the most complete of its many represented features.

Elements

OpenStreetMap data contains a large number of elements formatted in an XML file [60]. The four main elements in OSM data are as follows, quotations are from [60]:
Node: “A node defines a single geospatial point using a latitude and longitude. A third optional dimension, altitude can be recorded...” As a note, altitude does not seem to be widely recorded. “Nodes can be used to define standalone point features and also be used to define the path of a way... A node can be included as a member of relation and may have an associated Role.”

Way: A way is “an ordered list of between 2 and 2000 nodes. Ways can be used to represent linear features (vectors) or polygons (areas).” Polylines can be open or closed. “A way can be included as a member of relation and may have an associated Role. Ways of more than 2000 nodes will need to be split into two or more shorter ways. Ways may be ‘open’ (where they do not share a first and last node) or ‘closed’ where they do... Many roads... are described as open polylines.”

Relation: “A Relation consists of an ordered list of nodes, ways and sometimes also other relations as members of the new relation. The relation can have tags and each element can also optionally also have a defined role within the relation. A single element may appear multiple times in a relation and a relation can be included as a member of another relation.”

Tag: “A Tag should perhaps not be regarded as an “Element” in it’s own right, but rather it is a small unit of data attached to one of the above elements. A tag consists of two free format textual fields, a ‘key’ and a ‘value’... There are however many conventions on how individual features are best described.”

How these features are interpreted and used in this thesis is described in Chapter 4.

2.5 Computational Geometry

Computational Geometry is a subfield of Computer Science, which emerged in the 1970s, applying systematic computer science techniques such as the study of algorithms and data structures to geometric problems [17, p. 2]. Given its spatial focus, computational geometry lends itself well to the city design context, and has been applied to GIS in the past [17, p. 11]. There are two computational design techniques which are used in the thesis which are shown below. The first, enclosure detection (region extraction), is a complex, recent algorithm,
2.5.1 Region Extraction

The fitness function presented in this thesis is based on properties of the enclosures created by the nodes and edges of a city design’s road network. The fitness function is discussed in detail in 4.5. However, the fitness function requires these enclosures first be identified from that network. This is accomplished using a technique specified in An optimal algorithm for extracting the regions of a plane graph [41]. This technique extracts the regions of a plane graph in $O(n \log n)$ time, using $O(n)$ space, which is more efficient than previous approaches.

This technique takes as input a list of undirected edges, in arbitrary order. It operates in two phases: finding all wedges from a set of edges, followed by the extraction of regions from these wedges. This technique is shown in detail in Algorithms 10 to 12. In the algorithm, $(u_i, u_j)$ represents an undirected edge, while $\langle u_i, u_j \rangle$ represents a directed edge.

**Algorithm 10 Extract Regions**

1: function EXTRACT-REGIONS(edges)
2:    $wedges \leftarrow$ FIND-ALL-WEDGES(edges)
3:    $enclosures \leftarrow$ GROUP-WEDGES-INTO-REGIONS(wedges)
4:    return $enclosures$
5: end function

while rotation is a standard, basic technique, but is included for convenience.
Algorithm 11 Find All Wedges

1: function FIND-ALL-WEDGES(undirectedEdges)
2:     directedEdges ← empty list
3:     directedEdgesWithAngle ← empty list
4:     wedgesWithFlag ← empty list
5:     for all edge \((u_i, u_j)\) in undirectedEdges do
6:         add \((u_i, u_j)\) to directedEdges
7:         add \((u_j, u_i)\) to directedEdges
8:     end for
9:     for all edge \((u_i, u_j)\) to directedEdges do
10:         \(\theta\) ← the angle of \((u_i, u_j)\) with respect to the horizontal line passing through \(u_i\)
11:         add \((\langle u_i, u_j \rangle, \theta)\) to directedEdgesWithAngle
12:     end for
13:     sort directedEdgesWithAngle in ascending order using \(u_i\) and \(\theta\) as primary and secondary keys
14:     for all group in groups where group ≡ group of entries \((\langle u_i, u_j \rangle, \theta)\) with equal \(u_i\) do
15:         for all consecutive entries \((\langle u_i, u_j \rangle, \theta), (\langle u_i, u_k \rangle, \theta)\) in group do
16:             add a new wedge \((u_k, u_i, u_j)\) to wedgesWithFlag that is flagged as unused
17:         end for
18:         add a new wedge \((u_k, u_i, u_j)\) to wedgesWithFlag that is flagged as unused where \((\langle u_i, u_j \rangle, \theta), (\langle u_i, u_k \rangle, \theta)\) are the last and first entries in group
19:     end for
20:     return wedgesWithFlag
21: end function
Algorithm 12 Group Wedges into Regions

1: function Group-Wedges-Into-Regions(wedges)
2:     regions ← empty list
3:     currentRegionParts ← empty list
4:     sort wedges using $u_i$ and $u_j$ as primary and secondary keys
5:     while $\exists w \in W_1 \equiv$ the next unused wedge $(u_i, u_2, u_3)$ do
6:         $W_1 \leftarrow$ the next unused wedge such that $W_1 \equiv (u_i, u_2, u_3)$
7:         add $W_1$ to currentRegionParts
8:         flag $W_1$ as used
9:         $i \leftarrow 1$
10:        contiguous $\leftarrow$ false
11:        while !contiguous do
12:            $W_n \leftarrow W_{i+1} \equiv (u_{i+1}, u_{i+2}, u_{i+3})$ following $W_i \equiv (u_i, u_{i+1}, u_{i+2})$
13:            add $W_n$ to currentRegionParts
14:            flag $W_n$ as used
15:            contiguous $\leftarrow (u_{i+2} = u_1 \text{ and } u_{i+3} = u_2) \quad \triangleright W_{i+1}$ and $W_1$ are contiguous
16:            $i \leftarrow i + 1$
17:        end while
18:        add currentRegionParts to regions
19:        currentRegionParts $\leftarrow$ empty list
20:    end while
21:    return regions
22: end function

This algorithm extracts all of the regions formed from the edges. This includes the exterior region, which is not properly an enclosure. The extra step of removing the exterior region is discussed in 4.5.1, specifically, in Algorithm 37.

2.5.2 Rotation

Part of the expression mechanism involves rotating nodes in the phenotype (intersections in the road network). For convenience, the method of rotating a point around a given pivot
point is shown in Algorithm 13. This rotation method is a standard, basic operation, which is described in many sources, such as [25, p. 63].

**Algorithm 13 Rotate Point**

1: **procedure** Rotate-Point(point, rotationCenterPoint, angle)
2:   newX ← point’s x − rotationCenterPoint’s x
3:   newY ← point’s y − rotationCenterPoint’s y
4:   newXB ← (newX * cos(theta)) − (newY * sin(theta))
5:   newYB ← (newX * sin(theta)) + (newY * cos(theta))
6:   point’s x ← newXB + rotationCenterPoint’s x
7:   point’s y ← newYB + rotationCenterPoint’s y
8: **end procedure**

**2.6 Categorical Maps**

Categorical maps are images that provide information regarding properties of an underlying map by discretizing the data and grouping it by category. In this thesis, several categorical maps are created for a single phenotype, each map ‘colouring-in’ enclosures in the road network with different colours depending on the property considered in each map. More information on categorical maps are available in sources such as [12] and [27]. The use of categorical maps in CityBreeder is discussed in 4.5.

**2.7 Referenced Techniques**

In this section, several techniques are briefly summarized, which are used in various related works presented in the next chapter.

**2.7.1 Lindenmayer Systems**

Lindenmayer Systems (L-Systems), are parallel rewriting systems originally proposed by Astrid Lindenmayer in the paper *Mathematical Models for Cellular Interactions in Development I. Filaments with One-Sided Inputs* [46]. L-Systems can be described as follows: “The central concept of L-systems is that of rewriting. In general, rewriting is a technique for
defining complex objects by successively replacing parts of a simple initial object using a set of rewriting rules or productions” [66, p. 1].

L-Systems have become known as useful tools in a number of contexts, particularly in modelling plants, the use to which they were originally proposed [66, p. 1]. The influential book *The Algorithmic Beauty of Plants* discusses this application in depth [66]. They have also been used in many systems that involve branching, such as road network modelling, and are discussed in the next chapter.

### 2.7.2 Grammars

Grammars are used to transform an initial shape into a more complex shape. In the related works, *Shape Grammars* appear several times. Shape Grammars work with shapes which are “limited arrangement[s] of straight lines defined in a Cartesian coordinate system with real axes and an associated Euclidean metric” [72, p. 343]. Shape grammars are comprised of 4 elements [72, p. 347]: shapes, symbols, rules, and an initial shape. The rules are used to transform the initial shape into a more complex result.

Another grammar—*Split Grammar*—appears in one paper, and is discussed in Chapter 3 when discussing that approach.

### 2.7.3 Voronoi Diagrams

A Voronoi diagram contains structures which partition a space containing points, such that all of the area within a partition is associated with its nearest point. More formally, “Let S denote a set of n points (called site) in the plane. For two distinct sites \( p, q \in S \), the dominance of \( p \) over \( q \) is defined as the subset of the plane being at least as close to \( p \) as to \( q \),” [9].

### 2.8 Summary

In this chapter, background information was presented on specialized topics, which may be required to fully understanding the remainder of the thesis. Topics discussed include: city design, evolutionary computation, geographic information systems (GIS), computational geometry, categorical maps, and three additional techniques which are referenced in the
Related Work. With this information, the remaining chapters of the thesis should be more accessible.
Chapter 3

Related Work

3.1 Introduction

This chapter examines the state of the art of computer-based city design. Given the large number of procedural approaches that exist, a sample of those approaches that do not blend parent cities are examined first, followed by a the smaller number of those which do blend. Then, evolutionary approaches to this area are considered. Finally, for additional context, a sampling of procedural approaches that model other city features are briefly presented.

Throughout this chapter, critical attention is given to each approach, with a discussion of strengths and weaknesses, before concluding that no previous approach captures all of the requirements needed to answer the problem statement of this thesis: to enable the rapid, user-guided development of city designs based on the blending of multiple existing designs derived from real cities.

Each approach will be examined according to the requirements set out in the problem statement (1.2). These features were mentioned briefly in Chapter 1, and are discussed in more detail in Chapter 4. Each approach will be evaluated according to the following criteria:

- Scientific
- Rapid: Fast
- Rapid: Minimum Input Data Effort
- User-Guided: Any
- User-Guided: Blending
- City Design: Any Feature
- City Design: Road Network
Most of these terms have been defined in Chapter 1. However, regarding “Scientific”: almost all approaches considered in this chapter are scientific, with a notable exception. This exception is covered near the end of the chapter in the section on evolutionary approaches, and is discussed in detail. Also, several categories are divided between their strict requirement from the problem statement and a looser “any” version of that requirement to give credit to related approaches which come close to meeting it.

Specifically, the ‘user-guided’ requirement has been specified as requiring user-guidance during the blending process. A distinction is made between the presence of any user-guidance and user-guidance during blending specifically. Also previously defined, city designs are treated as consisting of road networks. Given the range of other city features discussed in 2.2.2, a distinction is made between approaches which model road networks, and those which model other city features. A distinction is also made between approaches which blend existing design road networks, and those which blend other features. Finally, a distinction is made between approaches which use real city data in any capacity, and those which use it during a blending process. How a related work may satisfy a looser requirement but not a strict one is explained on an individual basis.

3.2 Procedural Approaches

As discussed in Chapter 2, cities have many features. In the literature, different approaches model various combinations of these features. As discussed in the scoping section of Chapter 1, this thesis models the road networks of a city. Consequently, the procedural approaches discussed here tend to include road networks as at least one of their chosen features. For approaches which describe road network creation as only one phase of their pipeline, that phase will be explored in detail and its other phases will only be mentioned briefly.
All of the papers in this section and in the section on evolutionary approaches (3.3) share several common elements. Figure 3.1 shows one conceptual view of their common pipeline. It intentionally omits components of those papers’ approaches which deal with city features other than road networks.

![Diagram of conceptual pipeline for procedural approaches to city design](image)

Figure 3.1: Conceptual pipeline for procedural approaches to city design

Approaches in these two sections will be mapped onto this conceptual model. It is also worth noting that the components in Input and Feedback may not all be present in every model, and those that are not used will be shown in grey.

### 3.2.1 Surveys

In understanding procedural approaches to city design, it is useful to begin with a survey paper. *Modelling the Appearance and Behaviour of Urban Spaces* [79] is a comprehensive survey paper written by several prolific researchers in the field. It contains a large number of useful references, and discusses several aspects of city modelling including: procedural modelling, road network and layout modelling, building modelling, and rendering acceleration. The first two of these are particularly relevant to this thesis. Of particular interest, they identify a common ‘pipeline,’ which they suggest is used in most procedural approaches to city design. This pipeline is re-presented in Figure 3.2.
This pipeline accurately reflects many of the procedural techniques which are explored in this chapter which model both roads and buildings. The conceptual model presented in Figure 3.1 is similar, though it omits elements not pertaining to the road network generation, and adds user feedback.

Other less comprehensive surveys, which still offer some introductory insight into this field include: *A Survey of Procedural Techniques for City Generation* [42], which examines several procedural techniques and their use in city design. It considers each approach primarily from the perspective of the computer game industry, and analyzes each approach according to: realism, scale, variation, input, efficiency, control, and whether it is real-time. These metrics are appropriate for paper’s focus on applications to gaming, but are not used here as they do not consider many of the problem statement’s requirements such as blending or real data, and include other aspects which are not directly relevant. Additionally, *A Survey of Procedural Methods for Terrain Generation* [68] explores a number of procedural techniques for generating road networks and urban environments, but focusses on generating elevation data and simulating natural phenomena.

### 3.2.2 Non-Blending

Creating city designs by blending existing city designs has only been explored a few times, and these approaches are examined in the next section. The majority of procedural approaches to generating city designs do require some initial information, but not existing city designs. Samples of these approaches are explored in this section.
Procedural Modeling of Cities (CityEngine)

One of the earliest papers to apply procedural computer science techniques to city design is *Procedural Modeling of Cities*, by Parish and Müller [65], which remains influential as it continues to be cited frequently, and by the fact that the software has been acquired by the leading GIS software company—*ESRI* [24]—which has grown its capabilities.

*Procedural Modeling of Cities* introduces *CityEngine*, which creates city designs with roads and 3d textured building models. The pipeline they present appears in Figure 3.3. It is re-presented from [65, p. 302].

![Figure 3.3: Pipeline presented in Procedural Modeling of Cities](image)

In its pipeline, a number of image maps are first read into the program representing a
variety of information including: geographical maps (elevation, land, water, vegetation), and sociostatistical information (population density, zoning residential, commercial, mixed, street patterns, and height maps).

The road network is then generated using an extended L-System, and the enclosed regions are subdivided into lots. A building is created on each lot using another L-System which refines a polygon extruded from the lot base. Finally, the buildings are textured procedurally after the user has manually specified patterns in which multiple textures can be repeated and layered.

In creating the road network, the authors chose to divide the roads into two types: highways and streets. Highways connect areas of high population, while streets cover the areas between highways, following the dominant street pattern. In growing the highways, for each incomplete road segment, a number of rays within a given radius are extended outward, and the population density values sampled by the rays are weighted inversely proportional to their distance. The highway is then extended to the end of the best ray and the process repeated.

In creating streets, four basic street types are used:

- **Basic** - has no superimposed pattern. Rather, streets follow population density.

- **New York (rectangular grid)** - streets are placed in a grid according to an angle and specified values of the maximum width and height of a city block.

- **Paris (radial to center)** - roads follow a path around a central point that is specified or automatically calculated.

- **San Francisco** - longer streets exist on the same height level connected by short streets that follow a steep gradient.

The L-System used to create roads is modified to allow for loops, and progresses through three stages each time it adds a new road extension. First, a generic ideal candidate road extension is determined, but does not have any of its parameter values assigned. Next, parameters are proposed by the system in keeping with its global goals (street patterns and population density). Finally, local constraints (land/water/park boundaries, elevation, street crossings) are used to adjust or reject the parameters. This use of local constraints
acts as a repair mechanism, allowing for street segments ending near another road segment or intersection to be joined with it, or to shift roads nearing a body of water, resulting in roads which follow a shoreline.

The approach can be viewed according to the conceptual model previously articulated as shown in Figure 3.4.

In considering this approach with respect to the problem statement articulated in Chapter 1, it can be seen to succeed only partially with respect to the first requirement: rapid. Given the large amount of information initially required, and the relative difficulty of obtaining this information as discussed in Chapter 2, it does not satisfy rapid’s component: minimal input data effort.

Concerning rapid’s other component (fast), with CityEngine the “creation of the street graph [takes] less than 10 seconds, the division into lots and the creation of buildings approximately 10 minutes” [65, p. 307]. Given that this thesis is only concerned with the road network, it satisfies this requirement.

CityEngine is not user-guided in that it does not provide a feedback mechanism to guide design generation. To achieve a desired result, a user would need to execute the generation
process, examine the result, and manually modify the input maps (or even modify the intricate and technical L-System rules), and run the system again, repeating this process until a satisfactory result was obtained. Most of the examples in the paper pertain to artificial city designs. However, they do also apply the system to evolve a road network for Manhattan. It is unclear whether real data is used beyond land/water boundaries, but given the liberal definition for the ‘any’ version of using real city data, this qualifies.

A summary of its ability to satisfy the problem statement’s requirements is shown in Table 3.1.

<table>
<thead>
<tr>
<th>Scientific</th>
<th>Rapid: Fast</th>
<th>Rapid: Min Input Data Effort</th>
<th>User-Guided: Any</th>
<th>User-Guided: Blending</th>
<th>City Design: Any Feature</th>
<th>City Design: Road Network</th>
<th>Blends Existing Designs: Any</th>
<th>Blends Existing Designs: Road Network</th>
<th>Real City Data: Any</th>
<th>Real City Data: Blended</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>□</td>
<td>□</td>
</tr>
</tbody>
</table>

Table 3.1: Ratings for CityEngine

Image Analogies

Image Analogies is another early and influential paper that presents an interesting example-based technique applied to city design and other contexts [39]. Taking as input a source image (A) and a ‘filtered’ version (A’), the user can then present another image (B) and the system will produce a filtered version (B’) based on the mapping between A and A’.

The authors apply this technique to city design with the name Texture-by-numbers, where a city photograph is input as the target A’, and a simple RGB drawn image is input for A, which has been coloured simply, with the same colour in regions that correspond to regions in A’ which are of the same type. For instance, A might have yellow for residential neighbourhoods, blue for water, and red for commercial, all in the same places as those features exist in the photograph. Then, the user can present a new basic RGB drawn image (B) and the system will produce a new ‘photograph’ (B’) which appears realistic, having copy-and-pasted from appropriate regions from A’. For example, a new simply-coloured image can be drawn, and where yellow is drawn, new residential areas will appear, taken from the photographic source.

This technique maps onto the conceptual model as seen in Figure 3.5.
This approach is intuitive, fast, and yields plausible results, but does not fully satisfy the problem statement.

With regard to how fast the approach is, the authors state: “the performance of our algorithm is logarithmic with respect to the size (in pixels) of the training pair (due to using heuristic search techniques), and linear with respect to the size of the target. Although we have made use of several techniques to enhance performance (e.g., PCA, ANN), our algorithm is still rather slow, taking anywhere from on the order of tens of seconds to do simple texture synthesis, to a few minutes for the texture transfer examples, to a few hours for the artistic renderings . . .” [39, p. 6-7]. Given the city example they used likely falls under ‘texture synthesis’ (given its name Texture-by-numbers), this speed is sufficient given the definition in Chapter 1.

This approach does require user input in the form of RGB images. The complexity of these may vary greatly, but in the examples given in the paper, they appear simple enough that one could produce them quickly and easily.

The process is not explicitly user-guided. The system runs and produces a result and to guide the process, the input image B must be manually altered and input into the program again. Given the definition in Chapter 1, this does not qualify as user-guided.
Also, *texture-by-numbers* examples work with aerial photographs. This alone does not constitute a city design given the Chapter 1 definition.

In this approach, a new city design is created based on one existing city design, but the possibility of blending multiple parents is not discussed. While the paper does not mention such a possibility, it is possible that one could achieve this result by inputting as A two different city images concatenated side-by-side. A' would identify the different regions in each, and B could employ colours which are blends of similar regions in the two designs. As this was never investigated, it is not clear how well the system would perform in this manner, and is not considered achieved. However, as the system uses aerial photography, it can be seen to use some real city data. A summary of its ability to satisfy the problem statement’s requirements is shown in Table 3.2.

<table>
<thead>
<tr>
<th>Scientific</th>
<th>Rapid: Fast</th>
<th>Rapid: Min Input Data Effort</th>
<th>User-Guided: Any</th>
<th>User-Guided: Blending</th>
<th>City Design: Any Feature</th>
<th>City Design: Road Network</th>
<th>Blends Existing Designs: Road Network</th>
<th>Blends Existing Designs: Any</th>
<th>Real City Data: Any</th>
<th>Real City Data: Blended</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2: Ratings for *Image Analogies*

**Interactive Procedural Street Modeling**

*Interactive Procedural Street Modeling* creates a road network using tensor fields [13]. The system requires image maps as input: binary-valued water map, binary-valued park and forest map, height map, and a population density map. Using these, the system then proceeds through two phases: tensor field generation and street graph generation. The author’s pipeline also indicates a third category—3d geometry generation—but the authors admit this phase is not actually part of their program. Rather, it reflects taking the output from the second phase and inputting it to the modern commercial version of *CityEngine*. Their pipeline is re-presented in Figure 3.6.
Figure 3.6: Pipeline presented in *Interactive Procedural Street Modeling*

In the first phase, Tensor Field Generation, the system processes the input images to calculate the field’s eigenvectors and hyperstreamlines. The implementation also allows editing capabilities such as “combining individual basis fields, computing tensor values from boundaries, using a brush stroke interface, and rotating the field with noise,” [13, p. 103:2].

In the second phase, Street Graph Generation, streets are computed and created from the previously calculated hyperstreamlines. The authors chose to divide the roads into two types: *major roads* and *minor roads*. Major roads are “typically major business roads and local highways,” and minor roads are “usually residential and back roads,” [13, p. 103:2].

The second phase also includes many low-level editing capabilities, such as road segment manipulation, vertex manipulation, seed point creation, seed displacement, layered editing, and graph noise. The addition of new seed points triggers local regeneration.

How this approach maps onto the conceptual model articulated at the beginning of this section is shown in Figure 3.7.
The designs produced by the system appear plausible, but less realistic than many other approaches, including *Procedural Modeling of Cities*.

This approach satisfies some elements of the problem statement, but not all. With regard to speed: “a city with reasonable complexity can be modeled within five minutes, ... and [larger city segments] took about five minutes for the main layout, but required an additional thirty to sixty minutes to fine tune the details and to experiment with different designs.” While a final city can be generated in an hour, individual city designs take longer than one minute, and are consequently not considered fast.

This approach fails the minimal input data effort requirement as it requires 4 input maps, which would likely not be acquired easily in the manner described in Chapter 2. However, given the range of editing capabilities at both stages of city design generation, this approach is user-guided, though it does not blend multiple designs.

Finally, at one point in the paper, the authors state that existing road networks can be used “such as those obtained from Google Maps,” which then requires the user to indicate which regions are to be changed using the system. Additionally, the system is shown to generate road networks using land/water boundaries for multiple real cites, thus using real city data. A summary of its ability to satisfy the problem statement’s requirements is shown.
in Table 3.3.

<table>
<thead>
<tr>
<th>Scientific</th>
<th>Rapid: Fast</th>
<th>Rapid: Min Input Data Effort</th>
<th>User-Guided: Any</th>
<th>User-Guided: Blending</th>
<th>City Design: Any Feature</th>
<th>City Design: Road Network</th>
<th>Blends Existing Designs: Any</th>
<th>Blends Existing Designs: Road Network</th>
<th>Real City Data: Any</th>
<th>Real City Data: Blended</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.3: Ratings for *Interactive Procedural Street Modeling*

**Interactive Geometric Simulation of 4D Cities**

*Interactive Geometric Simulation of 4D Cities* simulates the growth of a 3d city over time, creating a “sequence of urban configurations” [85]. This approach creates a road network, subdivides lots and creates 3d building envelopes. The pipeline as articulated by the authors is re-presented in Figure 3.8.

Road network generation is heavily influenced by *Procedural Modeling of Cities*, adopting the taxonomy of *major* and *minor* roads, where major roads are grown first, with minor roads then grown to fill their interior regions. The main difference in the way roads are created, compared to the earlier approach, is that in this method the *order* in which road extensions are added is important, as the approach considers a city design’s growth over time. Similar to the four street patterns implemented in *Procedural Modeling of Cities*, three street patterns are implemented: *organic*, *grid*, and *radial*.

This approach requires a large amount of input data including: a height map, water map, forest map, “an initial urban layout configuration ranging from a single street up to a larger (potentially already existing city),” mathematically-defined land use type definitions, city center(s), growth center(s), percentage of new streets per year, average land price per year, patterns defining street expansion, land use percentage goals, construction parameter, and building generation rules. This approach maps onto the conceptual model as shown in Figure 3.9.
Figure 3.8: Pipeline presented in *Interactive Geometric Simulation of 4D Cities*
Figure 3.9: *Interactive Geometric Simulation of 4D Cities* mapped onto the conceptual model of procedural approaches to city design.
This approach produces plausible results, as did *Procedural Modeling of Cities*, but with the added feature of growth over time. For the problem statement investigated here, this added feature provides no advantage.

Regarding how rapid the approach is, the system runs at “about 1 second per time step” [85, p. 1]. However, it is not clear how many time steps are required to generate a developed final city design. Also, the paper does not spend more than one sentence discussing user feedback, but it mentions that streets can be manipulated at the low level during simulation, thus making it user-guided. While most of the paper deals with artificial data, an example near the end uses a map of Las Vegas as a starting point, thereby incorporating some real city data. These features are summarized in Table 3.4.

<table>
<thead>
<tr>
<th>Scientific</th>
<th>Rapid: Fast</th>
<th>Rapid: Min Input Effort</th>
<th>User-Guided: Any</th>
<th>User-Guided: Blending</th>
<th>City Design: Any Feature</th>
<th>City Design: Road Network</th>
<th>Blends Existing Designs: Any</th>
<th>Blends Existing Designs: Road Network</th>
<th>Real City Data: Any</th>
<th>Real City Data: Blended</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>?</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 3.4: Ratings for *Interactive Geometric Simulation of 4D Cities*

**Procedural City Layout Generation Based on Urban Land Use Models**

*Procedural City Layout Generation Based on Urban Land Use Models* generates city designs with roads and districts with associated land use types, [37]. It takes a number of inputs including: city diameter, continent (which determines district distribution), historical background (which determines districts in the city core), and number of highways. The paper is not clear on how many continents are supported or how historical background information is quantified, but discusses 3 types of Western European cities, 1 North American city and models 18 district types.

The system begins by generating terrain, and creates two circles demarcating the city’s core and outer limit. Next, it places preliminary highways around the core and connecting the two circles, determines a number of candidate district centers, selects the best ones, and then uses Voronoi partitioning to separate the districts. After applying noise, the approach from *Procedural Modeling of Cities* is applied to generate streets within each district. While the streets within a district may appear plausible, the overall city design does not appear convincing. While incorporating styles for different cities, it is unclear whether the rules
underlying these styles are created by hand to mimic them, or are based on real city data. This approach can be considered in relation to the conceptual model as shown in Figure 3.10.

![Procedural City Layout Generation Based on Urban Land Use Models](image)

**Figure 3.10: Procedural City Layout Generation Based on Urban Land Use Models** mapped onto the conceptual model of procedural approaches to city design

To summarize the system’s capabilities in relation to the problem statement, the “system can generate a large city in a few seconds” [37, p. 4], making it fast. Also, the required input data is a relatively small number of parameters which can be easily determined, meaning it requires minimal input data effort. However, the system is not user-guided and does not blend cities. This is summarized in Table 3.5.

<table>
<thead>
<tr>
<th>Scientific</th>
<th>Rapid: Fast</th>
<th>Rapid: Min Input Data Effort</th>
<th>User-Guided: Any</th>
<th>User-Guided: Blending</th>
<th>City Design: Any Feature</th>
<th>City Design: Road Network</th>
<th>Blends Existing Designs: Any</th>
<th>Blends Existing Designs: Road Network</th>
<th>Real City Data: Any</th>
<th>Real City Data: Blended</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.5: Ratings for Procedural City Layout Generation Based on Urban Land Use Models

**Interactive Design of Urban Spaces Using Geometrical and Behavioral Modeling**

In *Interactive Design of Urban Spaces Using Geometrical and Behavioral Modeling*, the relationship between city design and behaviour is explored while generating a road layout and 3d building geometry [82].
The system takes a number of input maps including: population, jobs, land value, road network, parcel shape, and building geometry. Behaviour is incorporated into the system through an agent-based approach which takes the input maps and is able to simulate changes to them over time, producing a more realistic design. Next, the road network is generated. The system uses 3 types of roads: highways “which are the highest capacity roads with limited access and carry inter-city traffic and, in larger cities, intra-city traffic;” arterials “which are lower capacity than highways, carry intra-city traffic and connect to local streets and highways;” and streets “which are local roads that route traffic from arterial roads to individual parcels” [82, p. 5].

The road network is generated by placing seeds for the different road types and growing them in a manner influenced by terrain and the input maps. The main population clusters are connected by placing initial seeds at those locations which are treated as arterial intersections. Further arterial roads are then generated from these initial intersections. Finally, street seeds are generated along arterial roads and used to generate streets. The pipeline offered by authors is re-presented in Figure 3.11.

![Figure 3.11: Pipeline presented in Interactive Design of Urban Spaces Using Geometrical and Behavioral Modeling](image-url)
This approach may be mapped onto the conceptual model as shown in Figure 3.12.

![Figure 3.12: Interactive Design of Urban Spaces Using Geometrical and Behavioral Modeling](image)

Limited user feedback is enabled through a paint-brush-like interface which allows the user to modify the imported maps, triggering regeneration.

The system is very fast, as it can generate city designs “containing up to 50,000 buildings, 3,000 km of roads, and 200 km² of area. The total interactive design process time, including several iterations of variable changes and modeling alterations, is just a few minutes on a standard desktop computer.” By contrast, the user will likely have to spend considerable time and effort collecting or producing all of the required input maps. The paper includes several results which appear realistic, but they do not seem to incorporate real city data. How this approach meets the requirements of the problem statement is summarized in Table 3.6.

<table>
<thead>
<tr>
<th>Scientific</th>
<th>Rapid: Fast</th>
<th>Rapid: Min Input Data Effort</th>
<th>User-Guided: Any</th>
<th>User-Guided: Blending</th>
<th>City Design: Any Feature</th>
<th>City Design: Road Network</th>
<th>Blends Existing Designs: Any</th>
<th>Blends Existing Designs: Road Network</th>
<th>Real City Data: Any</th>
<th>Real City Data: Blended</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.6: Ratings for *Interactive Design of Urban Spaces Using Geometrical and Behavioral Modeling*
Generating settlement structures: a method for urban planning and analysis supported by cellular automata

This approach applies cellular automata to the domain of city design. It models the spread of road networks and settlements, and explores the interaction between the two. It also includes an interesting analysis of different types of settlements, though these types do not appear widely in other papers in this field.

In creating a road network, the design area is discretized into two layered grids of different scales, and initial seeds are placed. At each time step, additional nodes are placed in neighbour cells with an edge between them. With some randomness, rules guide which of these neighbour cells will be selected and populated. Growth patterns are explored for a number of different settlement types. It may be mapped onto the conceptual model as shown in Figure 3.13.

Figure 3.13: Generating settlement structures: a method for urban planning and analysis supported by cellular automata mapped onto the conceptual model of procedural approaches to city design

In the paper, the system’s run time is not discussed, so it is not possible to ascertain how rapid it is. Also, while it seems that no input data is necessary, configuring the system with the appropriate parameters (including a networking parameter and centripetal and centrifugal forces parameters) and settlement types appears to be a highly technical process.
While this approach models several different kinds of settlement structures, it does not appear to incorporate real city data. How this approach meets the requirements of the problem statement is summarized in Table 3.7.

<table>
<thead>
<tr>
<th>Scientific</th>
<th>Rapid: Past</th>
<th>Rapid: Min Input Data Effort</th>
<th>User-Guided: Any</th>
<th>User-Guided: Blending</th>
<th>City Design: Any Feature</th>
<th>City Design: Road Network</th>
<th>Blends Existing Designs: Any</th>
<th>Blends Existing Designs: Road Network</th>
<th>Real City Data: Any</th>
<th>Real City Data: Blended</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>?</td>
<td>?</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.7: Ratings for *Generating settlement structures: a method for urban planning and analysis supported by cellular automata*

### 3.2.3 Blending

The papers discussed in the previous section are a sample of the large body of related work concerning (non-blending) procedural approaches to city design. In this section, procedural approaches are discussed which blend multiple existing designs together to generate new city designs.

**Interactive Modeling of City Layouts using Layers of Procedural Content**

Many procedural approaches to city design include some editing capabilities which allow the user to provide additional feedback, and make changes at a low or high level. *Interactive Modeling of City Layouts using Layers of Procedural Content* focusses exclusively on editing, providing a number of tools to manipulate city models that have been created through ESRI’s [24] *CityEngine* [23], working with the features: road network, land parcels, and parameter distributions for building generation [47].

The system implements several editing capabilities: drag, drop, insert, delete, rotate and scale, as well as the ability to combine information from multiple layers. Merging two layers is based on graph cuts, and their use of anchored assignments allows for local changes to be retained even if the rest of the model is regenerated. This approach can be mapped onto the conceptual model as shown in Figure 3.14.
Figure 3.14: *Interactive Modeling of City Layouts using Layers of Procedural Content* mapped onto the conceptual model of procedural approaches to city design

Though presented primarily as a paper introducing editing capabilities to procedural city design, its manner of doing so actually satisfies many aspects of the problem statement. The system can be seen in operation in a video [48], which demonstrates that it is *fast*, operating in real time.

As the system can use files from *ESRI’s CityEngine* [23] as input, it can be considered as requiring a minimum input data effort, because freely available mapping data (such as from *OpenStreetMap*) can be easily imported into *ESRI’s CityEngine*. While this means that one might be able to use the system with real data, it is unclear if this is done in the paper.

Unlike approaches previously discussed, the system is able to perform a limited blending of city designs. Through its use of layers and its ability to combine them, the properties of multiple city designs can be blended together to a limited extent. This can be clearly seen in the video. Discrete elements from different designs on different layers can be *mixed* together, producing a cut-and-paste effect. Additionally, the parameters for building heights can be *blended* together, producing a plausible effect. While this is a step towards the type of solution needed for the problem statement, it does not appear to *blend* the road network, as required in the Chapter 1 definition.

The use of multiple layers influenced the approach taken in this thesis, and is discussed
more in Chapter 4. How this approach meets the requirements of the problem statement is summarized in Table 3.8.

<table>
<thead>
<tr>
<th>Scientific</th>
<th>Rapid: Fast</th>
<th>Rapid: Min Input Data Effort</th>
<th>User-Guided: Any</th>
<th>User-Guided: Blending</th>
<th>City Design: Any Feature</th>
<th>City Design: Road Network</th>
<th>Blends Existing Designs: Any</th>
<th>Blends Existing Designs: Road Network</th>
<th>Real City Data: Any</th>
<th>Real City Data: Blended</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

Table 3.8: Ratings for *Interactive Modeling of City Layouts using Layers of Procedural Content*

**Interactive Example-Based Urban Layout Synthesis**

*Interactive Example-Based Urban Layout Synthesis* is a powerful approach related to the manipulation of ‘structured urban layout data,’ which is comprised of vector street data with corresponding aerial imagery [5].

Users provide structured urban layout fragments to the system which proceeds through two phases: Structure Synthesis, which generates a road network, and Image Synthesis, which fills the newly created parcels with aerial imagery.

In the Structure Synthesis (road network generation) phase, the input vector street data is processed, abstracting it so that rather than storing each street segment, only the computed intersections are retained, each with a number of attributes describing the area around the intersection and the streets which pass through it. This abstraction mechanism influenced the approach taken in this thesis and this influence is discussed more in the next chapter. The attributes computed and stored for each intersection include:

- Hierarchy levels of the two streets that could intersect that location (in processing the input data, streets are ordered according to a Gravelius ordering, which is often used in plant modelling. Highways start at level 0, and streets which branch off have an incremented hierarchy level)

- Average tortuosity (ratio of curve segment length to distance between endpoints) of each incident street

- Average area of nearby land parcels

- Statistical information for each of the two streets:
– Mean and variance of the distance between two consecutive intersection points
– Mean and variance of the angle between two consecutive street segments

Once the intersection points have been determined, the roads connecting them are generated using a random walk-based algorithm that is influenced by the attribute values. Tortuosity values are then used to subdivide the edges, creating curved polylines. City blocks are then generated as areas subdivided from the areas enclosed by the streets.

The Image Synthesis phase fills the lots with aerial imagery, but is beyond the scope of the current problem. For more information on this phase, the reader is directed to the original paper.

Of particular interest are the operations the system enables: _join, expand, and blend_. _Joining_ takes place when the user drops additional intersection points from one fragment next to another. The Structure Synthesis phase can then create streets that link the two sets of intersection points while preserving the properties of each. The _expand_ operation is not discussed in detail in the paper.

The _blend_ operation is what makes this approach particularly relevant as a potential answer to the problem statement. It can be seen in operation in a video [6]. Fragments can be placed one on top of the other, and the system is able to create roads which exhibit a blend of the two styles. It is a powerful tool, blending the properties of the two road networks at a deep level (as opposed to creating a _mixture_). This system can be mapped onto the conceptual model as shown in Figure 3.15.
This approach comes closer than previous approaches as a potential solution to the problem statement in many respects, but does not meet all criteria.

While the join operation allows the user to superficially mix two city designs, the blend operation enables blending in the deep sense as desired.

With regard to rapidity, the system is not fast. In the video, it is mentioned that the system takes approximately 10 minutes to generate a result. This exceeds the defined requirement of 1 minute per city design. However, if the aerial imagery is ignored, the amount of input data required is minimal (easily obtainable from a source such as OpenStreetMap).

The system contains several mechanisms to allow the user to influence the design, such as positioning the new segments, or expanding existing ones. However, the blending operation allows for very little user-guidance. Specifically, once the user has laid two city designs one on top of the other, they may only specify a blending coefficient which indicates how much of each is to be used, which is not much guidance. This approach uses real city data during blending. How this approach meets the requirements of the problem statement is summarized in Table 3.9.
This approach has been extended in several other papers. These papers share the same overall characteristics in regard to their ability to answer the problem statement, and so will only be briefly mentioned.

*Interactive Reconfiguration of Urban Layouts* takes this approach and adds several other high-level editing capabilities, including cut and paste, rotation, stretching, and ‘grow-around’ [3, 4]. The system adds the intermediate steps of computing boundaries and classifying each parcel by zoning type. When the layout is modified, individual tiles are deformed up to a specified threshold after which they will be removed, or if possible, substitute aerial imagery will be chosen.

*Visualization of Simulated Urban Spaces: Inferring Parameterized Generation of Streets, Parcels, and Aerial Imagery* pairs the approach taken in *Interactive Example-Based Urban Layout Synthesis* with simulation (using *UrbanSim* [83]), [81, 80]. This will grow the structured urban layout in a manner predicted by the simulation software.

While these extensions are intriguing, neither approach is more capable of solving the problem statement in this thesis.

### 3.3 Evolutionary Approaches

Evolutionary computation has been used to evolve solutions in many different problem domains. It has the ability to blend solutions as required by the problem statement. Though evolutionary computation has been applied successfully to different kinds of design problems, it has rarely been applied to city design. In this section, a direct application to city design is explored, along with several other closely-related applications.
3.3.1 City Generator

The closest application of evolutionary computation to city design appears in a thesis for a Master’s of Fine Art [73], and SIGGRAPH poster [74] which introduce City Generator: GIS Driven Evolution in Urban Simulation. Given the proximity of this approach to the approach proposed in this thesis (it is the only other application of EC to city design), it will be explored in detail. Even though the approach does not model roads, it is the closest application of evolutionary computation to city design that has been previously attempted.

In the approach, City Generator takes a number of 2d RGBA image maps as input, undergoes 3 phases, some of which involve GIS data, and produces a 3d city model which can then be imported into Maya [11] for high-resolution rendering, or to Microsoft XNA Game Studio [54] for real-time interaction.

The three phases of the system are:

- Phase I: Building generation (no GIS involved)
- Phase II: Automatic placement in the urban context (GIS involved)
- Phase III: Substitute proxy by procedural building methods (GIS involved)

One of the most significant problems with this approach is that it does not provide much detail, and is not conducted in a scientific manner. The evolutionary computation component is left as a ‘black box’ in many regards. These criticisms will be discussed further after describing the pipeline, which is re-presented from the author’s version, and shown in Figure 3.16.
Phase I consists of generating “ideal houses” in isolation to later be placed on a map in Phase III. These houses are developed by hand, by writing L-System expressions [73, p. 16].

In Phase II, the 2d maps input to the system are used to generate a ‘3D spatial occupancy (SO) model.’ How this data is used is not clear, and is discussed further later in this section.

In Phase III, buildings generated in Phase I are substituted into the model. It appears that this phase is optional as one example (‘Manhattan’, [73, p. 14]) suggests that pixel values from a map used in the previous phase can instead be used influence the height of boxes extruded from their locations.

The major strength of this approach is that it provides an interesting proof of concept for the application of evolutionary computation to city design. However, due largely to the fact that its investigation was not undertaken in a scientific manner, there are many weaknesses. The primary weakness is that it is ambiguous in many places and difficult to determine details. Given how ambiguous the language is at times, a number of quotations are provided
in the course of discussion below.

The evolutionary computation, the most important element of the approach, is left as a ‘black box’ in the author’s own pipeline. As discussed in Chapter 2, there are many forms of evolutionary computation. With City Generator, it is difficult to tell exactly which approach is taken. It appears to be a non-standard approach, with no reasons given for why the author departed from canonical forms. The author refers to his system as using a “Genetic Evolution (GE) engine” [73, p. 16], and does not discuss the actual representation of the genotype. The approach does make use of L-Systems, but their use appears to be confined to creating ‘ideal’ houses in Phase I, ending before the evolution begins.

Another problem with this approach is that it is unclear exactly which features are being represented. RGBA map images are input to the system, and given that the author discusses these channels separately and how they can be mapped to different features, it would appear that the system allows for up to 4 features to be modelled. However, many features are listed at various times including:

- Zoning [73, p. 18] or land use [73, p. 5]
- “Figure ground maps” [73, p. 5]
- Population [73, p. 18, 23]
- Transportation [73, p. 18]
- “Other spatial information” [73, p. 18]
- Family income [73, p. 23]
- DEM (digital elevation model) elevation data [73, p. 23]
- Time [73, p. 23]

It is unclear how these different maps are used in the pipeline, but it seems possible that the system allows up to 4 features and allows the user to determine how they are blended. One quotation elaborates on this:
For instance, a digital elevation model can be converted into a blue color TIFF image in GIS and later the blue color value can be read by City Generator to define the terrain elevation. A commercial zoning map can be converted into a red color map in GIS and later the red color can be read by City Generator to construct a series of commercial buildings. Each building has a unique height correlated with the land value. A vegetation map can be converted into a green color map and later the green color can be read by City Generator to grow various types of plants. Overlapping multiple 2D GIS maps forms the potential mixed-use zones and complicated urban features, which can be continuously configured in a simple image editing program based on the design criteria and then generate instant 3D results.” [73, p. 19]

It is unclear how much of this is implemented and how much is hypothetical. Another quotation makes the situation more ambiguous:

“For example, parent A is produced based on the population density data from the census bureau, while parent B is produced based on the land value data from real estate database, while parent C is produced based on the zoning map from local planning department. Groups of new urban forms are built up by mixing the gene [sic] of parent A, B, and C. The specific characteristic of each child is inherited from all parents through its voxels. Each voxel’s spatial value carried in parents is passed to their children’s related node with a randomly assigned weight by the “Blending Shape,” function in the 3D computer program.” [73, p. 15]

This casts further doubt on the possibility that the system is using a canonical form of EC. It suggests that specific children are created from individual colour maps, and individual features, which is unconventional. Another quotation suggests that the recombination portion of the evolution is also non-standard:

“For instance, 100 children are produced from each pair of “urban parents”. The first child is identical to parent A, while the 100th child is identical to parent B. The other 98 children are the mixture of parent A and B with a different weight combination.” [73, p. 15]
This suggests that *City Generator* creates 100 children each time two parents are selected, by exhaustively sampling a discretized version of the sample space. It also does not discuss how the blending is occurring; whether by averaging values, using a standard form of crossover, or some other variant. Additional information is given:

“First, five building units are modeled and imported into City Generator. Then I execute the breeding process and produce 3125 offspring in the first generation. This process is finished by an exhaustive combination of five original units’ genotype. From these 3125 samples, only five ideal spatial arrangement solutions are selected by reviewers and thus reserved as the genotypes for the next generation. To simulate the mutation, a nonlinear deformation node is evolved independently in the evolution and then explicitly added to the process to yield more complex layout potentials. In addition, a central courtyard, a negative volume, is also introduced into the evolution as a “void unit” and blended with the selected layouts.” [73, p. 12]

It is not necessarily a problem that the author diverges from standard forms of EC. However, since a scientific discussion of the advantages and disadvantages of doing so is missing, coupled with the general lack of detail, one cannot know if this is desirable.

The previous quotation also hints at a purely subjective fitness measure. The implication—of going through 3125 samples and choosing 5 ideal candidates—if correct, would seem to be far beyond a reasonable threshold for user fatigue. Another quotation reinforces the probability that a purely subjective interactive fitness measure is used:

“Because the complexity of fitness in urban design is far beyond the rules defined by computer algorithm [sic] such as cellular automation, evaluation and selection of the most-fit is exclusively operated by reviewers rather than artificial intelligence. In City Generator, the evaluation of fitness depends on the feasibility and functionality of a form in addition to its aesthetic value, which is subject to the reviewers’ decision.” [73, p. 19]

Adding to its characterization as unscientific, there is no discussion of the experimental parameter values, nor of quantitative results. The use of mutation is briefly hinted at while
explaining that the author “periodically stimulate[s] mutations in several members of the current population, and yield[s] a new candidate solution” [73, p. 18].

Additional confusion arises from a mis-use of terms. Throughout the paper, the ‘height’ of voxels is discussed, when in fact rectangles are being extruded from pixels, unlike actual voxels which have equal side lengths [73, p. 13].

This approach can be conceptualized as follows:

Figure 3.17: City Generator: GIS Driven Evolution in Urban Simulation mapped onto the conceptual model of procedural approaches to city design

Concerning the problem statement in this thesis, the fact that City Generator lacks roads is a problem. There is user-feedback through the interactive selection of individuals, but having to select 5 from 3125 means the approach is not fast. Given that a number of input image maps are required for data that is not easily accessible, the approach requires more than a minimum input data effort. It does not appear that real city data is used in blending. However, census data is used in the ‘spatial occupancy’ example, together with what appears to be a realistic land/water boundary.

This is summarized in Table 3.10.
While this is an interesting proof of concept for the application of evolutionary computation to city design, it does not answer the problem statement, and leaves room for the work done in this thesis.

3.3.2 Terrain

While evolutionary computation has not often been applied to city design, it has been applied to terrain generation which may be considered a city design feature in certain contexts. Though it is not directly applicable to the problem statement, some examples are briefly mentioned as they have had some influence on the work done in this thesis.

In Terrain generation using an Interactive Genetic Algorithm [84], their Auto Terrain Generation System (ATGS) is based on an interactive genetic algorithm. It evolves parameter values (represented as 8-bit binary strings) for a number of features including feature scale, spikiness, water level, sun direction, and clouds. These parameters are then input into the commercial terrain modelling and rendering software Terragen [69]. A population of 8 individuals are initially created with random parameter values, and the subsequent images rendered by Terragen are displayed to the user who chooses the top 3. These are then crossed-over and mutated and the evolutionary cycle continues. The system uses probability values for crossover and mutation of 0.7 and 0.1 respectively. The experiments conducted are qualitative in nature, with the authors using the system to try to generate scenes: “smooth low lying hills at sunset,” and “clear, tall and peaked mountains” [84, p. 5]. These qualitative targets influenced the design of the demonstrations which appear in Chapter 8. This approach does not use real city data. How this approach meets the requirements of the problem statement is summarized in Table 3.11.
Table 3.11: Ratings for Terrain generation using an Interactive Genetic Algorithm

Also worth mentioning briefly, Terrain Generation Using Genetic Algorithms also uses genetic algorithms to generate terrain [59]. Initially, the system takes a rough sketch from the user which indicates regional and terrain boundaries, together with a database of terrain data to create a rough terrain. The system operates in two phases, first evolving the terrain silhouette, then evolving the height field. The underlying representation uses floating point values, which encode an angle that influences the smoothness of the path between points along the silhouette’s boundary. Then, the enclosed regions in the silhouette are filled with sample data from the database.

Genetic algorithms have been used on other occasions to generate terrain, such as with Terrainosaurus [67]. In addition to genetic algorithms, genetic programming has been applied to terrain generation with approaches such as GenTP [29].

3.4 Other City Features

As this thesis focusses on the road network in city design, procedural approaches focussing exclusively on other city features are not explored in detail. As 3d building generation is discussed in the Future Work section in Chapter 9, a sample of procedural approaches to 3d building generation appear below.

Although L-Systems have been applied to 3d building design and architecture, grammars have found more traction, with considerably more research employing grammars in city design. The Palladian grammar uses a shape grammar to create villas using the Palladian style [71]. The language of the prairie: Frank Lloyd Wright’s prairie houses uses parametric shape grammars to create house designs in the Frank Lloyd Wright prairie house style [43]. More than the sum of parts: the grammar of Queen Anne houses applies shape grammar to generate house designs in the Queen Anne style [26], using separate grammars for its two phases: to generate floor plans, then to extrude the design into 3d. An Urban Grammar for the Medina of Marrakech: Towards a Tool for Urban Design in Islamic Contexts [19] uses
parametric urban shape grammar for the Zaouiat Lakhdar quarter of the Median of Marrakech in Morocco. It uses three sub-grammars: an urban grammar, negotiation grammar, and housing grammar.

Many other examples of grammars being applied to architecture exist, such as Procedural modeling of buildings, which applies shape grammar to 3d architecture [55]. Instant Architecture applies split grammars to architecture, requiring the user to specify some design goals, and uses a large database of over 250 rules as well as attributes and basic shapes [91].

In Example-Based Model Synthesis [51] and in Continuous Model Synthesis [52], a 3d closed polyhedral model is input by the user and the system analyzes the local relationships and connections between features in the model to create a wide range of output models which are unique but share properties with the original. In one example, the user inputs a typical office-building model, and the system generates a model of a downtown core, with a degree of uniqueness among each of the newly formed buildings. This approach can generate plausible looking urban areas as viewed from the ‘sky,’ but does not produce a realistic road network.

A similar approach is taken in Real-Time Procedural Generation of ‘Pseudo Infinite’ Cities [35] and Undiscovered Worlds—Towards a framework for real-time procedural world generation [36], which create buildings on the fly as the user moves over the city. Random buildings, created using seeds based on their locations, are generated by choosing a random shape for the building via extrusion and rotation.

Image-based Procedural Modeling of Facades takes a façade image and automatically subdivides it, determining a shape tree, then creates an accompanying 3d model [56]. Visual Editing of Grammars for Procedural Architecture extends shape grammar creation of buildings with a visual, 3d editing component [49].

Grammars and patterns more generally in urban design have also been investigated by Christopher Alexander in a number of works including A Pattern Language: Towns, Buildings, Construction [1], and A New Theory of Urban Design [2] which seek to describe patterns in design at multiple scales which combine to provide a sense of wholeness.

Many other approaches to the generation of 3d buildings and architecture exist, but only a sample is offered, as they will not be referenced outside of the Future Work section.
3.5 Summary

In this chapter, the state of the art of computer-based city design was examined, with procedural approaches (blending and non-blending), and evolutionary approaches considered. Each procedural and evolutionary approach was discussed and analyzed, and mapped against a common conceptual pipeline. Additionally, each approach was considered against each of the requirements in the problem statement. A summary of these results is shown in Table 3.12; some of the names are abbreviated for space. Additionally, the approach taken in this thesis—CityBreeder—is included in the table as the last entry. The following chapters will justify its inclusion and ratings.

An examination of the related work indicates that no previous approach has offered a complete solution to the problem articulated in this thesis. In the coming chapters, CityBreeder is introduced, and is shown to answer the problem statement.
<table>
<thead>
<tr>
<th>Paper</th>
<th>Scientific</th>
<th>Rapid: Past</th>
<th>Rapid: Min Input Data Effort</th>
<th>User- Guided: Any</th>
<th>User- Guided: Blending</th>
<th>City Design: Any Feature</th>
<th>City Design: Road Network</th>
<th>Blends Existing Designs: Any</th>
<th>Blends Existing Designs: Road Network</th>
<th>Real City Data: Any</th>
<th>Real City Data: Blended</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedural Modeling of Cities</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image Analogies</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>'?'</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Interactive Procedural Street Modeling</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>?</td>
</tr>
<tr>
<td>Interactive Geometric Simulation of 4D Cities</td>
<td>✓</td>
<td></td>
<td>'?'</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Procedural City Layout Generation Based on Urban Land Use Models</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interactive Design of Urban Spaces Using Geometrical and Behavioral Modeling</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generating settlement structures</td>
<td>✓</td>
<td></td>
<td>'?'</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interactive Modeling of City Layouts using Layers</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>?</td>
</tr>
<tr>
<td>Interactive Example-Based Urban Layouts Synthesis</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>City Generator: GIS Driven Evolution in Urban Simulation</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Terrain generation using an Interactive GA</td>
<td>✓</td>
<td></td>
<td>'?'</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>CityBreeders</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 3.12: Ratings Summary
Chapter 4

System Design and Implementation

4.1 Introduction

In this chapter, a system named CityBreeder is presented and analyzed, which enables the rapid, user-guided development of city designs based on the blending of multiple existing designs derived from real cities, thereby answering the problem statement.

In this chapter, the system is discussed, including its pipeline, which consists of two phases. Also discussed are its components, including its representation, genetic operators, and fitness function. Each of these is discussed and analyzed in terms of design and implementation. The design describes its general capabilities, while the implementation discusses any additional constraints added or encountered in implementing it for use in conducting the experiments which appear in the next few chapters.

The system proceeds through two main phases:

1. Discovering City Genes - Phenotypes created from imported data initially lack a corresponding genotype. This phase discovers those genes through evolution.

2. Breeding Cities - Allows a user to interactively create new city designs by breeding together those which have been imported and have then had their genes discovered.

The system contains a process to import data from OpenStreetMap, which produces a phenotype (discussed in 4.2.1). However, most of the system’s components are related to genetic programming and its evolutionary process. These components may be broadly grouped into: representation, genetic operators, and fitness function.

The representation reflects the ontological structure of an individual. In particular, it includes the genotype, phenotype, and expression mechanism for transforming a genotype into a phenotype. Its representation consists of the following components:

- **Genotype representation (4.3.1)** - a multi-layered structure consisting of quadtrees which spatially reflect city design features.
• **Phenotype representation** (4.3.2) - a road network consisting of collections of nodes and edges representing roads.

• **Expression mechanism** (4.3.3) - a mechanism for expressing a given genotype into a phenotype. It is a process with 3 phases: Grow, Rotate, and Zoom.

Genetic operators are required by the evolutionary process to combine, modify and select individuals within a population. Each operator in this system falls into one of three categories:

• **Crossover** (4.4.1) - creates new individuals by combining the genetic material from two parents. Crossover operators include: Basic Crossover and Weak Context-Preserving Crossover.

• **Mutation** (4.4.2) - alters the genotype of a single individual with some randomness. Mutation operators include: Expand Leaf to Quadtree Mutation, Compress Leaves-Only Quadtree to Leaf Mutation, Gaussian Leaf Value Mutation, Permute Leaves-Only Quadtree Mutation, and Permute Quadtree Mutation.

• **Selection** (4.4.3) - chooses individuals from a population at two points during the evolution process. Selection operators include: Rank Selection, Uniform Selection, and Interactive Selection.

The last main component of the system is the fitness function, which is presented in 4.5. The fitness function used in Discovering City Genes employs computational geometry and categorical mapping techniques, and operates on the enclosures within a given road network. Breeding Cities does not actually require a fitness function as neither selection mechanism requires a fitness value, and Breeding Cities experiments do not have a specific target pre-specified against which to measure similarity. This design decision, and how objective fitness values might be incorporated into phase 2 of the pipeline are discussed more in 4.4.3.

### 4.1.1 Symbol Tables

Below are two tables of symbols representing various values in the system. They are offered here for quick reference, but each is discussed in the section in which it is introduced. Table 4.1 shows symbols which have been embedded into the design of the system and influence its
operation. Table 4.2 lists the user-facing system parameters. In addition to these parameters, depending on which phase of the pipeline is being executed, additional parameters will be available, including the following for Discovering City Genes: target city, initial parents, and choice of selection mechanisms. Breeding Cities includes the additional parameter: target city side length.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
<th>Represents</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>1</td>
<td>Minimum Quadtree Depth</td>
<td>4.3.1</td>
</tr>
<tr>
<td>β</td>
<td>0.001</td>
<td>Standard Street Length</td>
<td>4.2.1</td>
</tr>
<tr>
<td>γ</td>
<td>4000</td>
<td>Categorical Map Side Length</td>
<td>4.5.1</td>
</tr>
<tr>
<td>ζ</td>
<td>8</td>
<td>Number of Designs to Display to User</td>
<td>4.4.3</td>
</tr>
<tr>
<td>ι</td>
<td>100</td>
<td>Difference Map Side Length</td>
<td>4.5.1</td>
</tr>
<tr>
<td>κ</td>
<td>0.5</td>
<td>Stop at Terminal Probability</td>
<td>2.3.2</td>
</tr>
<tr>
<td>μ</td>
<td>0.335</td>
<td>Scale X Fitness Weight</td>
<td>4.5.1</td>
</tr>
<tr>
<td>ν</td>
<td>0.335</td>
<td>Scale Y Fitness Weight</td>
<td>4.5.1</td>
</tr>
<tr>
<td>ξ</td>
<td>0.33</td>
<td>Angle Fitness Weight</td>
<td>4.5.1</td>
</tr>
<tr>
<td>σ</td>
<td>1.41422</td>
<td>Expression Zoom Amount</td>
<td>4.3.3</td>
</tr>
</tbody>
</table>

Table 4.1: Fixed Values Symbol Table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Γ</td>
<td>Population Size</td>
</tr>
<tr>
<td>Δ</td>
<td>Number of Generations</td>
</tr>
<tr>
<td>Θ</td>
<td>Crossover Probability</td>
</tr>
<tr>
<td>Ω</td>
<td>Ratio of Basic to Weak-Context Preserving Crossover</td>
</tr>
<tr>
<td>Ξ</td>
<td>Expand Leaf to Quadtree Mutation Probability</td>
</tr>
<tr>
<td>Π</td>
<td>Compress Leaves-Only Quadtree to Leaf Mutation Probability</td>
</tr>
<tr>
<td>Σ</td>
<td>Gaussian Mutate Leaf Value Mutation Probability</td>
</tr>
<tr>
<td>Τ</td>
<td>Permute Leaves-Only Quadtree Subtrees Probability</td>
</tr>
<tr>
<td>Φ</td>
<td>Permute Quadtree Subtrees Probability</td>
</tr>
<tr>
<td>Ψ</td>
<td>Angle Contiguity Window Value</td>
</tr>
</tbody>
</table>

Table 4.2: Parameters Symbol Table

4.2 Pipeline

A pipeline is a process consisting of multiple components where the output from one component acts as input to another. The system presented in this thesis, CityBreeder, contains a pipeline with two phases. This general pipeline used may be visualized as shown in Figure 4.1.
Figure 4.1: *CityBreeder* Pipeline

This process may also be alternatively simplified as shown in Figure 4.2.
In the first phase, *Discovering City Genes*, genotypes are evolved for phenotypes which have been imported from OSM data. This phase proceeds automatically, and may take longer than the second phase as it will likely require a larger population and more generations.

After multiple city designs have been imported and their genes discovered, phase two of the pipeline may be executed with *Breeding Cities*. In this phase, the user can breed new city designs from existing ones.

Both phases employ evolution, though in slightly different ways. Each phase can be seen as mapping onto the basic evolutionary loop articulated in 2.3 in Algorithm 1, as shown in Algorithms 14 and 15.

Both phases use genetic programming, but the first phase is automated while the second requires user interaction. This pipeline can be mapped onto the conceptual model used in Chapter 3 as seen in Figure 4.3.
Algorithm 14: Discovering City Genes Mapped onto the Evolutionary Computation Pipeline

1: initialize population with random candidate solutions using Ramped Half-and-Half Initialization
2: evaluate fitness of each individual using Categorical Map Similarity Fitness
3: while termination condition (number of generations reached) not satisfied do
   4:   select parents using Rank Selection or Uniform Selection
   5:   recombine pairs of parents using 2 forms of Crossover
   6:   mutate resulting offspring using 5 forms of Mutation
   7:   evaluate fitness of offspring using Categorical Map Similarity Fitness
   8:   select individuals for next generation using Rank Selection or Uniform Selection
4: end while

Algorithm 15: Breeding Cities Mapped onto the Evolutionary Computation Pipeline

1: initialize population with random candidate solutions using Prepopulated Initialization
2: evaluate fitness of each individual is not required
3: while termination condition (user chooses to quit) not satisfied do
   4:   select parents using Interactive Selection
   5:   recombine pairs of parents using 2 forms of Crossover
   6:   mutate resulting offspring using 5 forms of Mutation
   7:   evaluate fitness of offspring is not required
   8:   select individuals for next generation using Uniform Selection
9: end while
This mapping is a bit difficult to follow as it shows the system as being both user-guided and not user-guided. For clarity, this mapping can be split up, with each phase mapped separately onto the conceptual model. These are shown in Figures 4.4 and 4.5.

Examining the conceptual model with Discovering City Genes, evolutionary parameters and OSM data are first input in the Input portion. Next, within the Generate City Design block, the genotype is determined from the OSM data, and an automatic GP component determines the city design’s phenotype. The final output is the imported city segment’s design comprised of both genotype and phenotype.
Figure 4.4: *CityBreeder’s Discovering City Genes* phase mapped onto the conceptual model presented in Chapter 3

In the mapping of the Breeding Cities phase, evolutionary parameters are input along with existing city designs from the Discovering City Genes phase. The user-guided GP component is represented by the Generate City Design ↔ Feedback loop. The final output is a new city design based on existing designs.

Figure 4.5: *CityBreeder’s Breeding Cities* phase mapped onto the conceptual model presented in Chapter 3
4.2.1 Import

In order for new designs to be created from existing ones that are based on data derived from real cities, data derived from real cities must be imported into the system. Though not perhaps its own phase, this mechanism is used to import OSM data, yielding a phenotype, which will be used as a target in the Discovering City Genes phase which will search for a corresponding genotype.

OSM data is formatted as XML, and its structure and elements are described in 2.4.2. The importing occurs in three phases:

- Import Phase I: Parse OSM XML
- Import Phase II: Convert to Intermediate Representation
- Import Phase III: Convert Intermediate Representation to Phenotype

**Import Phase I: Parse OSM XML**

For the import process to begin, an OSM file must be selected, and useable boundary information (minimum and maximum latitude and longitude) provided. In early iterations of the system, there was no distinction between ‘useable’ bounds and the bounds in the OSM file. However, it is not clear how much data outside of these bounds is exported by OpenStreetMap, and imports from early iterations were left with many ‘non-areas’ around the design’s border, where enclosures could not be found as connected, out-of-bounds nodes and edges were missing. Allowing the system to require the user to specify the desired ‘useable’ bounds separately minimizes this problem as a larger area can be exported from OpenStreetMap, intentionally leaving some extra buffer space. This is discussed more in 4.5.

In Phase I, The OSM file is opened, and its data is parsed and recorded in a number of data structures:

- XML metadata
- OSM metadata
- Bounds (Minimum Longitude, Maximum Longitude, Minimum Latitude, Maximum Latitude)
• An array of nodes
• An array of ways
• An array of relations

In this phase, the data is not modified in any way, merely input and recorded.

Import Phase II: Convert to Intermediate Representation

In phase 2, the bounds information is retained, and arrays of arrays of highway and street segments are created. Later, in phase 3, these will be combined into a single road network with one array of nodes, and one of edges.

These are recorded as arrays of arrays as that is essentially what is defined by the ‘array of ways’ in the source data, except that here the ‘way’ type is eliminated and only arrays of arrays of nodes are retained. Additionally, a distinction is made between highway and street segments in this phase because early iterations of the system included a distinction between highways and roads, and it further allows for such a distinction to be more easily re-introduced in future work if desired.

In this phase, an array of nodes is created for each way, and is then added to one of two arrays of arrays depending on whether the way contained one of the following tags:

• **highway tags** = \{“motorway”, “primary”\}

• **street tags** = \{“secondary”, “tertiary”, “residential”, “service”\}

This list is not exhaustive, and OpenStreetMap’s taxonomy appears to allow for many variations. An exhaustive list was not found during research. This list appears to work well for Ottawa and Paris, yet misses some segments for New York. However, the New York segment was still used with this import process as it provided an additional challenge for the system (discussed further in 7.4). Future work might determine a more complete list of OpenStreetMap road type tags and incorporate them.

Separating this phase from the next is reasonable from the perspective of modularizing the system. Generally speaking, this phase assembles the raw data into structures approximating those in the phenotype, while the next phase further refines and remaps that data. More
simply, this phase affects how the data is recorded, while the next affects how it is interpreted. This separation makes it slightly easier to enable future work to change either.

**Import Phase III: Convert Intermediate Representation to Phenotype**

In this phase, a phenotype is created from the intermediate representation produced by the previous phase.

Phenotypes are interpreted by the system as having boundaries from (0,0) to (1,1). The first step in this phase is to use the specified useable bounds to remap all of the node coordinates. This will leave all coordinates within the useable bounds with new coordinate values between 0 and 1. Nodes which are out-of-bounds will be imported, but will not affect the phenotype. They are imported as their presence may help the fitness function in the identification of enclosures.

Because the useable bounds will be used to remap the coordinates, useable bounds should be selected which produce a square area. If not, the phenotype may appear stretched.

From these useable bounds, two important values are determined for the imported design: *city side length* and *maximum tree depth*. *City side length* is recorded as the smaller of the two sides of the import (the minimum of the absolute values of differences of the latitude and longitude values).

*Maximum tree depth* is calculated from the *city side length* and *standard street length*. The *standard street length* is defined as:

\[ \beta \equiv 0.001. \]

This value was chosen after a visual examination of several cities in OpenStreetMap. \( \beta \) must be assigned a small enough value such that streets evolved can be as small and ‘dense’ as those in any given target city. However, assigning it a value that is too small has been seen in early tests of the implementation to increase runtime substantially.

*Maximum tree depth* is then calculated as the number of times *city side length* is divided by 2 to be smaller than \( \beta \). Both *city side length* and *maximum tree depth* are later used in expressing a genotype into a phenotype.

The final step in this phase iterates over both arrays of arrays from the previous phase, adding each edge to the new phenotype’s list of edges. This will add edges, and in the process also adds nodes which are not already stored. In other words, it adds without duplication.
4.2.2 Discovering City Genes

In Discovering City Genes, genetic programming is used to evolve a genotype which corresponds with an imported phenotype. This stage is necessary as the next phase—Breeding Cities—requires the genotypes in order for breeding to occur.

This phase is largely automatic, requiring a user to select an imported design, and perhaps adjust evolutionary parameters slightly. However, this may not be necessary as ‘reasonable’ values are pre-set, and are discussed further in 6.2.

This phase may take longer than the Breeding Cities phase as it may require a larger population or larger number of generations depending on the complexity of the target design. The GP component follows the evolutionary loop presented in Algorithm 1, finding individuals with high fitness which correspond visually with the target phenotype. Strictly speaking, these genes are not the actual genes of the target city design, but are rather the genes of an approximation of the target design. The goal of this phase of the pipeline is to minimize the error on this approximation, so that the genotype ultimately discovered can be considered with the highest possible confidence to be the genotype for the target design.

4.2.3 Breeding Cities

In this phase, new city designs can be created by breeding together designs derived from existing cities. Specifically, genotypes corresponding to real city designs, as discovered in Discovering City Genes, are used as parents in an interactive evolutionary process. This process is largely identical to the one used in the previous phase, except for several minor differences. Specifically, the differences can be summarized as follows:

- A fixed population size of 8
- Round one of selection is Interactive Selection
- Round two of selection is Uniform Selection

The fixed population size is related to the question of user fatigue. Other interactive evolutionary work encountered during research tended to use or suggest that 8 or 9 items should be shown to the user at a time. One such example which presented 8 individuals at a time, and was discussed in the related work is Terrain Generation Using an Interactive
Genetic Algorithm [84]. Ideally, this allows the user to see all individuals on a screen at once without becoming overwhelmed.

Round one of selection in the evolutionary process is Interactive Selection. At the beginning of each generation, the user is prompted to select members of a parents pool for that generation. Next, from those parents, 16 children are produced, from which 8 are uniformly selected to survive to the next generation. Uniform selection is used because there are no objective fitness values. It is possible to use objective fitness values, if a predefined target is selected, and was implemented in this manner in an early iteration of the system before being removed. This possibility is discussed further in Future Work (9.2).

4.2.4 Evolutionary Computation and Genetic Programming

Thus far, this section has described the pipeline used in the system, but did not explain the components in detail. This was done to give a high-level view, and also because many of the same evolutionary components are used in both phases. Specifically, genetic programming is featured in both phases, with a few differences in purpose, parameters, and operators. The next few sections describe each of the components of the evolutionary system in detail.

4.3 Representation

As is standard in genetic programming, the genetic representation consists of a genotype, phenotype, and genotype-to-phenotype expression mechanism. The genotype consists of layered quadtrees which spatially reflect features of a road network constituting a city design, and the phenotype consists of lists of nodes and edges. The expression mechanism interprets the genotype spatially in its three phases: grow, rotate, and zoom.

The representation is designed for the city design context, but also has several generally useful properties, and might be extended to other domains. This will be explored further in Future Work (9.2).
4.3.1 Genotype

Design

The genotype is designed as a collection of n quadtrees, each one representing a layer in the genotype. Each may be mapped to a different city feature. The leaves in each layer can hold a value between specified minimum and maximum values. During phenotypic expression, each layer is interpreted as representing that feature in a spatial manner. For example, if a layer consists of a quadtree with depth 1 (a quadtree with 4 leaves), and the northwest cell contains a particular value, then phenotypic expression will interpret the northwest quadrant of the city design being evolved as ‘possessing’ that trait. In other words, the layers will be ‘stretched’ over the city when being expressed in creating a phenotype. This expression process is discussed in 4.3.3, and examples of this spatial relationship are clearly illustrated in the examples in Chapter 5.

Implementation

The genotype is implemented with one type of layer, which contains floating point values. However, the design allows for other types of layers. For instance, early iterations of the system had both float and integer layers.

The genotype is implemented with 3 layers: Scale X, Scale Y, and Angle. Each layer reflects certain properties of enclosures within a phenotype’s road network. Enclosures were discussed earlier in 2.5.1, and are discussed more later in this chapter when presenting the fitness function in 4.5.

Visualization

The genotype is never shown to the user, but is shown many times throughout the experimentation chapters. The genotype may be visualized in many different ways. Examples below show visualizations of several of genotypes. Perhaps the version closest to how it is stored on the computer is to view a genotype textually. One of the longer individuals found in one of the later experimentation chapters had the following genotype (leaf values are shown with two decimal places, but in the implementation are stored as floats in Java [64]):

**Scale X**: \{1.00, 0.99, \{\{1.00, 1.00, 1.00, 0.99\}\}, 1.00, 0.99, \{\{1.00, 1.00, 1.00, 1.00\}\}, 0.99,
This approach does not reflect a spatial understanding of a genotype, and can be hard to read and understand. An alternative is to display each genotype layer as a tree. One possible layer of another genotype is shown in Figure 4.6.

Figure 4.6: Tree Visualization of a Genotype Layer
This approach is more readable, and reflects the hierarchical nature of the layer, but does not yet reflect a spatial understanding. If each subtree is interpreted as a spatial quadrant of a square map (as indeed it is during expression), then another genotype might be visualized as shown in Figure 4.7.

![Figure 4.7: Example Square Genotype](image)

The leaf values are between 0 and 1, which may be mapped to colour intensity to create colour images. Given that there are 3 layers, another genotype with maps for scale x (red), scale y (green), and angle (blue), might appear as shown in Figure 4.8.

![Figure 4.8: Example Square Genotype (RGB)](image)

This can be useful for showing values at a glance, especially with deep trees where it may be difficult to read off the leaf values as in the previous example. Additionally, since there are currently 3 layers, each may be used as a colour channel in an RGB image. A single image created from the genotype previously shown would appear as shown in Figure 4.9.
While this approach may produce interesting images, it is difficult to extract information out of it quickly at a glance. As a result, it will not be used. Rather, in this thesis, when genotypes are presented, they will be shown as square images with text as in Figure 4.7. If the trees are so deep that the numbers in the images are hard to read, accompanying separate-RGB images (as in Figure 4.8) may be provided for a quick visual reference.

Initialization

The evolutionary process operates agnostically with respect to how the initial population is created. The automatic GP component in the Discovering City Genes phase uses Ramped Half-and-Half Initialization, while the GP component in Breeding Cities uses Prepopulated Initialization.
Ramped Half-and-Half Initialization

Design
Ramped Half-and-Half Initialization is used in Discovering City Genes to create the initial population in a random fashion as this positions the system from an initial position of high diversity. It is used in a manner similar to the standard approach which is discussed in 2.3.2.

Implementation
Minor variations from the standard form appear in the implementation, including how the ramps are calculated. For ease of use (and as rapid: minimal input data effort is a desired feature in the problem statement), rather than having the user specify the specific tree depths for the ramps, they are calculated for the following depths:

- $1/5$ maximum tree depth
- $2/5$ maximum tree depth
- $3/5$ maximum tree depth
- $4/5$ maximum tree depth

Each ramp generates $1/4$ of the initial population. These ramps were chosen because they provide a wide range of possible depth values. Depth 0 is not used since simple examples tested during early development showed it produced designs which were excessively simple. Additionally, initializing to the full depth was not added as it is typically characteristic of GP to have the average tree depth increase over time. The maximum tree depth used in this process is determined during import as discussed in 4.2.1.

As discussed in Chapter 2, Ramped Half-and-Half Initialization makes use of Full and Grow algorithms. One non-standard value was used in the implementation of Grow: $\text{Minimum quadtree depth} \equiv \alpha := 1$. This value was initially implemented as early iterations of the system produced poor results and had simple trees which appeared to be part of the cause. The presence of this minimum value will not prevent shallow, simple trees from being created if they have high fitness, but merely creates an initial population with a larger degree of tree
depth and complexity. In fact, some of the fittest solutions in later experimentation chapters have at least some layers with simple trees.

**Prepopulated Initialization**

Prepopulated Initialization is used in the Breeding Cities phase to create the initial population. Unlike Ramped Half-and-Half which creates a randomized initial population, Prepopulated Initialization creates a population that shares characteristics with specified parents.

**Design**

Multiple existing designs are brought into the evolutionary process at its beginning by creating an initial population comprised of offspring which are already a blend of the parents. The initial population is created by crossing-over the two parents and adding the offspring to the initial population, repeating for a specified population size.

**Implementation**

Prepopulated Initialization allows 2 parents to be specified. The algorithm is shown in Algorithm 16.

**Algorithm 16 Prepopulated Initialization**

1: function PREPOPULATED-INITIALIZATION(populationSize, twoParents)
2:     loop populationSize times
3:         randomize the ordering of twoParents
4:         twoChildren ← Crossover(twoParents)
5:         add twoChildren’s first child to initialPopulation
6:     end loop
7:     return initialPopulation
8: end function
4.3.2 Phenotype

Design

A phenotype contains road networks comprised of nodes and edges. Nodes consist of longitude and latitude values, and edges contain head and tail node references.

Implementation

In the implementation, the phenotype contains a single road network, and spans a square area with boundaries spanning from (0,0) in the north west to (1,1) in the south east. It may contain nodes which are out-of-bounds: these will not affect the appearance of the phenotype but may affect enclosure detection which occurs during fitness evaluation.

Phenotypes are created by importing OSM data (4.2.1), and also through the expression of a genotype. An example imported phenotype is shown in Figure 4.10.

4.3.3 Expression Mechanism

As discussed in Chapter 2, the process by which a phenotype can be created from a genotype can be direct or indirect. Since the genotype’s values are used as input to a separate development process in creating a phenotype, this expression mechanism is indirect. The expression mechanism is also explored later in Chapter 5 through a number of examples.

The expression mechanism consists of three main phases: Grow, Rotate, Zoom. However, there are other minor steps, and the full algorithm is shown in Algorithm 17.

The algorithm calculates a minimum street length value from city side length and standard street length. City side length is determined during import, and later assigned from target to candidate individuals. The standard street length, $\beta$, discussed in 4.2.1, has a value of 0.001.

It should also be noted that some of the algorithms reference Growth-Check, which checks if a given vertex could be the seed of a subsequent extension. This is accomplished by checking if the given point plus some small amount on the x and y is located within the ‘filled’ area recorded in totalArea.
**Grow**

The first phase of expression, Grow, begins in the top left corner of the phenotype and creates a rectangular enclosure by extending along the x and y in relation to values on the scale x and scale y layers of the genotype. The extension is modified to account for possible overlap with previously generated extensions, new seed points are added for the new vertices and the process is repeated until the usable area is filled. The algorithm for this phase is shown in Algorithm 18. The process that determines the extension amounts is illustrated in Figures 4.11 and 4.12.

Note, this process should not be confused with Ramped-Half-and-Half Initialization’s ‘Grow’ method of creating a genotype. This is referred to as ‘Grow-Expression’ in algorithms to avoid confusion.
Algorithm 17 Genotype-to-Phenotype Expression

1: function Expression(genotype, citySideLength)
2:     minStreetLength ← β ÷ citySideLength
3:     scaleXMap ← GENERATE-MAP(genotype's scale x layer, γ, minStreetLength)
4:     scaleYMap ← GENERATE-MAP(genotype's scale y layer, γ, minStreetLength)
5:     angleMap ← GENERATE-MAP(genotype's angle layer, γ, minStreetLength)
6:     phenotype ← empty phenotype
7:     GROW-EXPRESSION(phenotype, scaleXMap, scaleYMap, minStreetLength)
8:     ROTATE(phenotype, angleMap, minStreetLength)
9:     ZOOM(phenotype)
10:    return phenotype
11: end function

Rotate

Once Grow has finished, a valid city design will have been created, but all line segments will be at angles parallel to either the x or y axis. The Rotate phase introduces more variety regarding these angles. This is accomplished by determining the proposed rotation for every enclosure, grouping them into rotation layers, applying rotation to each layer, and finally combining them. This process is explicated in Algorithm 22.

Zoom

Following rotation, it is possible that there will be ‘non-areas’ in the corners of the map. This arises because enclosures previously there have been rotated to other locations. This produces somewhat unrealistic gaps in the corners of the design. This is compensated for during the Zoom phase. The algorithm for this phase is shown in Algorithm 28.

The expression zoom amount is defined as σ := 1.41422, which is approximately equal to the amount a square which exactly fit the usable area would need to be scaled up by (from its center) to avoid having any non-areas around its edges, when rotated at any angle. This is illustrated in Figure 4.13.

This may not always eliminate all non-areas, as rotation occurs at the ‘center of mass’ for a given angle group, and it is possible that the grow phase extends slightly beyond the
Algorithm 18 Grow (Expression)

1: procedure Grow-Expression(phenotype, scaleXMap, scaleYMap, minStreetLength)
2:   allSeeds ← empty list of nodes
3:   toRemove ← empty list of nodes
4:   toAdd ← empty list of nodes
5:   totalArea ← empty area
6:   add node(0, 0) to allSeeds
7:   while allSeeds is not empty do
8:     empty toRemove
9:     empty toAdd
10:    for all nodes in allSeeds do
11:      if !GrowthCheck(node) then
12:        add node to toRemove
13:      else
14:        width ← Lookup-Growth-Extension(scaleXMap, node)
15:        height ← Lookup-Growth-Extension(scaleYMap, node)
16:        if (width ≤ 0) or (height ≤ 0) then
17:          add node to toRemove
18:          continue
19:        end if
20:        proposedArea ← area from node to (node’s x + width, node’s y + height)
21:        proposedArea ← proposedArea – totalArea
22:        totalArea ← totalArea + proposedArea
23:        Add-Proposed-Area(phenotype, proposedArea, toAdd)
24:        add node to toRemove
25:      end if
26:    end for
27:    remove all members of toRemove from allSeeds
28:    add all members of toAdd to allSeeds
29:  end while
30: end procedure
Algorithm 19 Lookup Growth Extension

1: function LOOKUP-GROWTH-EXTENSION(scaleMap, node)
2:   \( m \leftarrow \beta \div (\text{citySideLength} \times \sigma) \quad \triangleright\text{zoom 'density'} \)
3:   \( a \leftarrow \text{LOOKUP-AVERAGE-VALUE}(scaleMap, node, \text{sideLength}) \quad \triangleright\text{avg. value for area} \)
4:   \( \text{error} \leftarrow \text{sideLength} \div (m + ((1 - a) \times (1 - m))) \)
5: return \( \text{sideLength} \) such that \( \text{error} \approx 0 \quad \triangleright\text{this was implemented using binary search} \)
6: end function

Algorithm 20 Lookup Average Value

1: function LOOKUP-AVERAGE-VALUE(scaleMap, node, sideLength)
2: return the average value over the 'surface' of a square on the scaleMap beginning at node and extending to (node's x + sideLength, node's y + sideLength)
3: end function

Algorithm 21 Add Proposed Area

1: procedure ADD-PROPOSED-AREA(phenotype, proposedArea, toAdd)
2: for all lineSegments along proposedArea's path do
3:   add lineSegment to phenotype's nodes and edges
4: if GROWTHCHECK(lineSegment's tail) then
5:   add lineSegment’s tail to toAdd
6: end if
7: end for
8: end procedure
Algorithm 22 Rotate

1: procedure Rotate(phenotype, minStreetLength)
2:   Find-Enclosures(phenotype)
3:   Identify-Enclosure-Angles(phenotype, angleMap)
4:   \[ \text{listOfEnclosures} \leftarrow \text{empty list} \]
5:   add all of phenotype’s enclosures to \text{listOfEnclosures}
6:   sort \text{listOfEnclosures} according to each member’s quantizedProposedAngle
7:   enclosuresHistogram \leftarrow \text{empty list}
8:   enclosuresHistogramValues \leftarrow \text{empty list}
9:   Create-Angular-Enclosure-Histograms(\text{listOfEnclosures},
   \text{enclosuresHistogram}, \text{enclosuresHistogramValues})
10:  \text{rotationLayers} \leftarrow \text{Get-Rotation-Layers}(\text{enclosuresHistogram},
   \text{enclosuresHistogramValues})
11:  Apply-Rotation(\text{rotationLayers})
12:  Combine-Rotation-Layers(\text{rotationLayers})
13:  Find-Enclosures(phenotype)
14: end procedure

Algorithm 23 Create Angular Enclosure Histograms

1: procedure Create-Angular-Enclosure-Histograms(\text{listOfEnclosures},
   \text{enclosuresHistogram}, \text{enclosuresHistogramValues})
2:   for all enclosures in \text{listOfEnclosures} do
3:     enclosure’s area \leftarrow \text{calculate enclosure’s area}
4:     index \leftarrow \text{enclosure’s quantizedProposedAngle}
5:     if index = 90 then
6:       index \leftarrow 0
7:     end if
8:     add enclosure to \text{enclosuresHistogram} at \text{index index}
9:     \text{enclosuresHistogramValues index index} \leftarrow \text{enclosuresHistogramValues index index} + \text{enclosure’s area}
10:   end for
11: end procedure
Algorithm 24 Get Rotation Layers

1: function GET-ROTATION-LAYERS(enclosuresHistogram, enclosuresHistogramValues)
2:   allRotationLayers ← empty list
3:   biggestArea ← −1
4:   while biggestArea ≠ 0 do
5:     rotationLayer ← the enclosures from the n consecutive cells in enclosuresHistogram that correspond to the largest sum of values in corresponding n cells in enclosureValues where the size of n = Ψ
6:     add rotationLayer to allRotationLayers
7:     for all index in n do
8:       clear enclosuresHistogram's cell at index
9:     end for
10:   end while
11:   return allRotationLayers
12: end function

Algorithm 25 Apply Rotation

1: procedure APPLY-ROTATIONS(rotationLayers)
2:   for all layers in rotationLayers do
3:     for all nodes in the layer's enclosures do
4:       ROTATE-POINT(node, layer's center of mass, layer's average angle converted to radians)
5:     end for
6:   end for
7: end procedure
Algorithm 26 Combine Rotation Layers

1: procedure COMBINE-ROTATION-LAYERS(\text{rotationLayers}, \text{phenotype})
2: \hspace{1em} if rotationLayers is size 1 then
3: \hspace{2em} for all edges in rotationLayers’ single layer do
4: \hspace{3em} add edge and edge’s nodes to phenotype if they are not already contained (no
5: \hspace{3em} intersection checks required)
6: \hspace{1em} end for
7: \hspace{1em} else
8: \hspace{2em} for all layers in rotationLayers do
9: \hspace{3em} for all edges in layer do
10: \hspace{4em} add edge and edge’s nodes to phenotype if they are not already contained
11: \hspace{4em} (intersection checks and corrections required)
12: \hspace{3em} end for
13: \hspace{1em} end for
14: end if
15: end procedure

Algorithm 27 Identify Enclosure Angles

1: procedure IDENTIFY-ENCLOSURE-ANGLES(\text{phenotype}, \text{genotype})
2: \hspace{1em} for all enclosures in phenotype do
3: \hspace{2em} averageAngle $\leftarrow$ 90 $\ast$ (weighted average value of the intersecting regions of
4: \hspace{3em} genotype’s angle layer projected over the phenotype)
5: \hspace{2em} enclosure’s quantizedAngle $\leftarrow$ averageAngle rounded to the nearest whole num-
6: \hspace{3em} ber
7: \hspace{1em} end for
8: end procedure
east or south boundaries. However, this phase significantly reduces the presence of these ‘non-areas.’

**Algorithm 28** Zoom

```plaintext
1: procedure Zoom(phenotype)
2:     for all nodes in phenotype do
3:         node’s longitude ← ((node’s longitude − 0.5) * σ) + 0.5)
4:         node’s latitude ← ((node’s latitude − 0.5) * σ) + 0.5)
5:     end for
6: end procedure
```

### 4.4 Operators

A number of genetic operators are used which fall into the categories: crossover, mutation, and selection. Most of these follow the standard forms presented in Chapter 2, while others are more tailored to the representation. Extra attention is given where the implementation diverges from the standard form.
4.4.1 Crossover

Crossover is described in detail in 2.3.2. The two crossover mechanisms used—Basic Crossover and Weak Context-Preserving Crossover—are implemented in their canonical forms, with one minor difference as the genotype consists of multiple layers. In the implementation, this crossover process is applied to each layer independently. Therefore, the scale x layers of the parents are crossed over, selecting swap points according to that operator’s rules. Next, the remaining two layers repeat the process without influence of what occurred on either other layer.

The overarching crossover algorithm may be seen in Algorithm 29. Within the calls to Basic-Crossover and Weak-Context-Preserving-Crossover the operations occur on each layer separately.

4.4.2 Mutation

Mutation is described in 2.3.2. In that section, general forms of Permutation Mutation and Gaussian Value Mutation are described. These are implemented in a manner tailored to genotype representation. Additionally, operators to expand and compress the quadtrees are employed.

The five mutation operators used in the system are:
• Expand Leaf to Quadtree Mutation

• Compress Leaves-Only Quadtree to Leaf Mutation

• Gaussian Mutate Leaf Value Mutation

• Permute Leaves-Only Quadtree Subtrees Mutation

• Permute Quadtree Subtrees Mutation

Unlike crossover which operates on each layer of the genotype independently, mutation chooses one layer randomly within which to operate. Additionally, unlike mutation which is often found in related work which chooses one single mutation operator to use on an individual in that instance of mutation, in CityBreeder, each mechanism has a probability of operating on an individual during a single course of mutation. These two characteristics were chosen because early iterations of the system featured fewer, more powerful mutation operators, which produced poor results. As a result, the decision was made to ‘weaken’ the mutation operators, add more, and allow them all to operate with some probability. Much
better results were immediately seen. The general mutation algorithm can be viewed in Algorithm 30.

**Expand Leaf to Quadtree Mutation**

Expand Leaf to Quadtree Mutation randomly selects a leaf from a randomly selected layer in the genotype and replaces it with a quadtree consisting of 4 new leaf nodes each of which possess the value of the original leaf. It serves the purpose of adding depth and ‘detail’ in the tree. Its algorithm is shown in Algorithm 31.

**Compress Leaves-Only Quadtree to Leaf Mutation**

Compress Leaves-Only Quadtree to Leaf Mutation randomly selects a quadtree whose subtrees are all leaves from a randomly selected layer in the genotype and replaces it with a new leaf with a value equal to the average of the original quadtree’s 4 leaves. It serves the purpose of shortening or subtracting ‘detail’ in the tree. Its algorithm is shown in Algorithm 32.

**Gaussian Mutate Leaf Value Mutation**

Gaussian Mutate Leaf Value Mutation randomly selects a leaf node from a randomly selected layer in the genotype and adjusts its value according to a Gaussian function. It serves the purpose of introducing new leaf values, which is not possible through crossover.
Algorithm 30 Mutation
1: function Mutation(genotype)
2:     if a random value $\in [0.0..1.0] < \Xi$ then
3:         Expand-Leaf-to-Quadtree-Mutation(genotype)
4:     end if
5:     if a random value $\in [0.0..1.0] < \Pi$ then
6:         Compress-Leaves-Only-Quadtree-to-Leaf-Mutation(genotype)
7:     end if
8:     if a random value $\in [0.0..1.0] < \Sigma$ then
9:         Gaussian-Mutate-Leaf-Value-Mutation(genotype)
10:    end if
11:    if a random value $\in [0.0..1.0] < \Upsilon$ then
12:       Permute-Leaves-Only-Quadtree-Subtrees-Mutation(genotype)
13:    end if
14:    if a random value $\in [0.0..1.0] < \Phi$ then
15:       Permute-Quadtree-Subtrees-Mutation(genotype)
16:    end if
17: end function

Algorithm 31 Expand Leaf to Quadtree Mutation
1: procedure Expand-Leaf-to-Quadtree-Mutation(genotype)
2:     selectedLayer $\leftarrow$ randomly selected layer in genotype
3:     selectedLeaf $\leftarrow$ randomly selected leaf node in selectedLayer
4:     newQuadtree $\leftarrow$ new Quadtree with 4 leaves with selectedLeaf’s value
5:     replace selectedLeaf with newQuadtree
6: end procedure
Algorithm 32 Compress Leaves-Only Quadtree to Leaf Mutation

1: procedure COMPRESS-LEAVES-ONLY-QUADTREE-TO-LEAF-MUTATION(genotype)
2:     selectedLayer ← randomly selected layer in genotype
3:     selectedLeavesOnlyQuadtree ← randomly selected quadtree whose subtrees are all leaves in selectedLayer
4:     newLeaf ← new leaf with value equal to the average of selectedLeavesOnlyQuadtree’s four leaf values
5:     replace selectedLeavesOnlyQuadtree with newLeaf
6: end procedure

alone. Its algorithm is shown in Algorithm 33.

The Gaussian function was used with 3 standard deviations being equal to 0.5, as it produced ‘small’ changes to the value on the order of 0.1 or 0.2 fairly regularly.

Algorithm 33 Gaussian Mutate Leaf Value Mutation

1: procedure GAUSSIAN-MUTATE-LEAF-VALUE-MUTATION(genotype)
2:     selectedLayer ← randomly selected layer in genotype
3:     selectedLeaf ← randomly selected leaf node in selectedLayer
4:     selectedLeaf’s value ← selectedLeaf’s value + NEXTGAUSSIAN(mean ← selectedLeaf’s value, standardDeviation)
5:     replace selectedLeaf with newQuadtree
6: end procedure

Permute Leaves-Only Quadtree Subtrees Mutation

Permute Leaves-Only Quadtree Subtrees Mutation randomly selects a quadtree whose subtrees are all leaves from a randomly selected layer in the genotype and randomly permutes its subtrees. Its algorithm is shown in Algorithm 34.

Permute Quadtree Subtrees Mutation

Permute Quadtree Subtrees Mutation randomly selects any quadtree from a randomly selected layer in the genotype and randomly permutes its subtrees. Its algorithm is shown in Algorithm 35.
Algorithm 34  Permute Leaves-Only Quadtree Subtrees Mutation

1: procedure PERMUTE-LEAVES-ONLY-QUADTREE-SUBTREES-MUTATION(genotype)  
2:   selectedLayer ← randomly selected layer in genotype  
3:   selectedLeavesOnlyQuadtree ← randomly selected quadtree whose subtrees are all  
   leaves in selectedLayer  
4:   randomly permute selectedLeavesOnlyQuadtree’s subtrees  
5: end procedure

Algorithm 35  Permute Quadtree Subtrees Mutation

1: procedure PERMUTE-QUADTREE-SUBTREES-MUTATION(genotype)  
2:   selectedLayer ← randomly selected layer in genotype  
3:   selectedQuadtree ← randomly selected quadtree in selectedLayer  
4:   randomly permute selectedQuadtree’s subtrees  
5: end procedure

These mutation operators are not too far removed from their more canonical forms where present, but are adapted to the genotype representation.

4.4.3 Selection

As mentioned in Chapter 2, selection occurs at two points: Selecting who gets to mate and who gets to survive.

Rank and Uniform Selection

Rank and Uniform Selection are used in Discovering City Genes, and Uniform selection is also used in Breeding Cities. Both are implemented in their canonical forms, which are described in 2.3.2.

Interactive Selection

Interactive selection is used in Breeding Cities during round one of selection, and allows the user to select individuals to be added to a parent ‘pool’ which will be bred together to create the 2n intermediate offspring population (Uniform Selection is then used in the following
Design

A number of phenotypes are shown the user who may then select from them. At least one must be selected to continue.

Implementation

Eight individuals are shown to the user, who may then click to select or deselect them. Selections are highlighted with a blue border. Once all selections have been made, the user may 'select and continue' to the next generation, or 'select and quit' which will halt evolution at the end of that generation. As mentioned in 4.2.3, 8 designs are shown to the user.

An example of the window shown to the user, with two selections having been made, is shown in Figure 4.14.

![Interactive Selection Window Displayed to User](image)
4.5 Fitness Evaluation

A method of evaluating a given phenotype’s fitness is an important part of an evolutionary system. In Discovering City Genes, a fitness function is used which employs computational geometry tools to analyze the properties of the enclosures in the phenotype.

As discussed earlier in examining the pipeline in 4.2, during the Breeding Cities phase a fitness value is not required as it is not used by either selection mechanism and there is no objective target against which to determine similarity. The reasons for this design decision and possible future work which would include an objective fitness measure are discussed in that section as well as in 9.2.

4.5.1 Categorical Map Similarity

The remainder of this section will discuss Categorical Map Similarity, which is the fitness function used in Discovering City Genes.

Design

The fitness function is based on the comparison of categorical maps which describe features of the enclosures within the phenotype’s road network.

Put simply, the system requires a method of comparing the phenotypes. Rather than comparing the nodes and edges (intersections and streets) directly, the system examines all of the enclosures within the road networks, determines properties of these enclosures, which are recorded (or painted) in categorical maps. Finally, these categorical maps are compared on a pixel-by-pixel basis for similarity. Once similarity values have been calculated for each feature (layer), a fitness value may be determined as a weighted average of these values.

Implementation

In CityBreeder, there is one categorical map for each layer of the genotype, mapping that feature. The enclosures are all identified for a phenotype, oriented bounding boxes are fitted to each, and properties of these bounding boxes are calculated (scale x, scale y, and angle). Each value is used to determine a colour which fills the enclosure on the corresponding categorical map. Next, the three categorical maps for a given phenotype can be compared
against the three categorical maps for the target phenotype and a fitness score calculated based on similarity. This process is summarized in Algorithm 36.

Examples showing the categorical maps for an imported city design are displayed in Figure 4.15 with and without the enclosures’ oriented bounding boxes visible. Categorical maps are created with a specified side length. In the experiments, this \textit{categorical map side length} \( \equiv \gamma := 4000 \).
Figure 4.15: Example Categorical Maps
In calculating the fitness of a candidate design, difference maps may be extracted which reflect the pixel comparisons made in Algorithm 39. While these maps are not typically created during normal execution (the difference values are merely aggregated), they can be created if desired (with a debug flag). Example difference maps are shown in Figure 4.16. Dark pixel values indicate high similarity.

![Example Difference Images](image)

Figure 4.16: Example Difference Images

Because this pixel-by-pixel comparison can take a long time depending on the implementation, the categorical maps are initially shrunk down. In the experiments presented in the following chapters, a difference map side length $\equiv \iota := 100$ was used, creating 100x100 pixel approximations. The selection of this value reflects a trade-off between speed and accuracy, both of which are called for by the current problem. The effects of this choice of value is explored in the experimentation chapters. Future work might increase this value at higher generations; as the population begins to show less diversity this higher value will allow smaller differences to be detected.

Similarity values are calculated for each of the three features (layers), and a final fitness value is calculated using a weighted average of them. These weights should sum to 1, and in the implementation, the following values are used to give approximately equal weight to all three features:

- **Scale X Fitness Weight** $\equiv \mu := 0.335$
- **Scale Y Fitness Weight** $\equiv \nu := 0.335$
• **Angle Fitness Weight** \( \equiv \xi := 0.33 \)

One note for the Categorical Maps Fitness algorithm is that pixel values should be normalized between 0 and 1. Whether an extra normalization step is required will depend on the software implementation. Also of note is that angle values ‘wrap’ when considering similarity. That is, the similarity of angle values 2 and 87 will be the same as that of 2 and 7.

**Algorithm 36** Fitness Function

1: `function Fitness-Function(candidatePhenotype, targetPhenotype)`
2: `    Find-Enclosures(candidatePhenotype)`
3: `    Generate-Categorical-Maps(candidatePhenotype)`
4: `    fitness ← Categorical-Maps-Fitness(candidatePhenotype, targetPhenotype)`
5: `    return fitness`
6: `end function`

**Algorithm 37** Find Enclosures

1: `procedure Find-Enclosures(candidatePhenotype)`
2: `    candidatePhenotype’s enclosures ← Extract-Regions(candidatePhenotype’s edges)`
3: `    remove candidatePhenotype’s largest enclosure`
4: `end procedure`

4.6 **How the System Answers the Problem Statement**

The problem statement is to enable the rapid, user-guided development of city designs based on the blending of multiple existing designs derived from real cities. The system design and implementation discussed in this chapter answers this problem statement in principle. Each component is addressed:

• **Rapid** - regarding minimal user input: the import and Discovering City Genes are automated and do not require much input beyond the imported city. The speed of the system will be discussed in Chapter 8.
Algorithm 38 Generate Categorical Maps

1: procedure GENERATE-CATEGORICAL-MAPS(phenotype)
2:   phenotype’s scaleXMap ← blank image (size $\eta \times \eta$), filled solid magenta
3:   phenotype’s scaleYMap ← blank image (size $\eta \times \eta$), filled solid magenta
4:   phenotype’s angleMap ← blank image (size $\eta \times \eta$), filled solid magenta
5:   for all enclosures in phenotype do
6:     valueX ← Min(enclosure’s orientedBoundingBox’s scaleX value, 1.0)
7:     valueY ← Min(enclosure’s orientedBoundingBox’s scaleY value, 1.0)
8:     valueA ← enclosure’s orientedBoundingBox’s scaleX value ÷ 90
9:     drawingColourX ← colour with intensity $\propto$ valueX
10:    drawingColourY ← colour with intensity $\propto$ valueY
11:    drawingColourA ← colour with intensity $\propto$ valueA
12:    draw enclosure to phenotype’s scaleXMap filled with drawingColourX
13:    draw enclosure to phenotype’s scaleYMap filled with drawingColourY
14:    draw enclosure to phenotype’s angleMap filled with drawingColourA
15:   end for
16: end procedure
Algorithm 39 Categorical Maps Fitness

1: function CATEGORICAL-MAPS-FITNESS(candidateRaw, idealRaw)
2:     candidate ← candidateRaw shrunk to $i \times i$
3:     diffScaleX ← diffScaleY ← diffAngle ← 0
4:     numIdealPixels ← 0
5:     for $x \leftarrow 0 \rightarrow i$ do
6:         for $y \leftarrow 0 \rightarrow i$ do
7:             pIX ← ideal’s pixel at $(x, y)$ on scaleXMap
8:             pIY ← ideal’s pixel at $(x, y)$ on scaleYMap
9:             pIA ← ideal’s pixel at $(x, y)$ on scaleAMap
10:            pCX ← candidate’s pixel at $(x, y)$ on scaleXMap
11:            pCY ← candidate’s pixel at $(x, y)$ on scaleYMap
12:            pCA ← candidate’s pixel at $(x, y)$ on scaleAMap
13:               if $pIX \neq$ magenta then $\triangleright$ ideal’s pixel is not blank
14:                  if $pCX \neq$ magenta then $\triangleright$ candidate’s pixel is not blank
15:                     diffScaleX ← diffScaleX + $|pIX - pCX|$
16:                     diffScaleY ← diffScaleY + $|pIY - pCY|$
17:                     diffAngle ← diffAngle + $\min(|pIA - pCA|, |pIA - pCA + 1|)$
18:                  else
19:                     diffScaleX ← diffScaleX + 1
20:                     diffScaleY ← diffScaleY + 1
21:                     diffAngle ← diffAngle + 1
22:               end if
23:           end if
24:       end for
25:     end for
26:     return CALCULATE-WEIGHTED-SIMILARITY(diffScaleX, diffScaleY, diffAngle, numIdealPixels)
27: end function
Algorithm 40 Calculate Weighted Similarity

1: function \textsc{Calculate-Weighted-Similarity}(\texttt{diffScaleX}, \texttt{diffScaleY}, \texttt{diffAngle}, \texttt{numIdealPixels})
2: \hspace{1em} \texttt{diffScaleX} ← \texttt{diffScaleX} ÷ \texttt{numIdealPixels}
3: \hspace{1em} \texttt{diffScaleY} ← \texttt{diffScaleY} ÷ \texttt{numIdealPixels}
4: \hspace{1em} \texttt{diffAngle} ← \texttt{diffAngle} ÷ \texttt{numIdealPixels}
5: \hspace{1em} \texttt{similarityScaleX} ← 1 − \texttt{diffScaleX}
6: \hspace{1em} \texttt{similarityScaleY} ← 1 − \texttt{diffScaleY}
7: \hspace{1em} \texttt{similarityAngle} ← 1 − \texttt{diffAngle}
8: \hspace{1em} \texttt{weightedSimilarityFitness} ← (\mu \times \texttt{similarityScaleX}) + (\nu \times \texttt{similarityScaleY}) + (\xi \times \texttt{similarityAngle})
9: \hspace{1em} return \texttt{weightedSimilarityFitness}
10: end function

- **User-Guided** - in Breeding Cities, the user is able to guide the evolutionary process by selecting parents at each generation through Interactive Selection.

- **Development of City Designs** - a genetic representation (layered quadtree genotype, phenotype with nodes and edges, expression mechanism) was presented that is suited to the city design context. Additionally, the representation lends itself well to further extensions (as will be discussed in 9.2), which is essential for the city design context should users desire additional city features.

- **Based on the Blending of Multiple Existing Designs** - Evolutionary computation lends itself naturally to blending individuals. Prepopulated Initialization ensures that the existing designs (parent cities) influence the design process from the very beginning.

- **Derived from Real Cities** - The system has a mechanism to import data exported from \textit{OpenStreetMap}, yielding phenotypes based on real city data. The genotypes for these cities are discovered, and they are blended to create new city designs.
4.7 Summary

In this chapter, CityBreeder was presented and it was shown how its design and implementation answer the problem statement in principle. Its two-phase pipeline was presented along with its import and evolutionary components, including genotype and phenotype representations, genetic operators, selection mechanisms, and fitness.

In the following chapters, experiments and examples are shown which demonstrate properties of the system and answers the question of how well it answers the problem statement. The system will be tested with simple examples as well as shown operating on real city data.
Chapter 5

Expression Examples

5.1 Introduction

The preceding chapter introduced CityBreeder’s design and implementation, and argued that it provides a complete solution to the problem statement, thereby answering it in principle. This is the first of four ‘experimentation’ chapters which examine how well it answers the problem statement. This chapter contains examples and analysis whereas the next three include full evolutionary runs from both phases of CityBreeder’s pipeline (Chapter 9 also features examples).

In this chapter, several examples of genotype-to-phenotype expression are presented. These serve to illustrate how the expression mechanism works as well as help to determine how expressive the genetic representation is. To answer the problem statement effectively, the expression mechanism should be expressive enough to reasonably capture and reflect features of real city road networks.

Genotype and phenotype examples are shown with three images each. Each phenotype image is zoomed out to show more of the grid, as portions will extend beyond the usable area, which is demarcated by a green box. These cropped areas are shown to better demonstrate the expression, even though they are not visible to the user during evolution, and do not play a role in determining fitness (except in identifying enclosures).

Each phenotype figure contains three sub-figures showing the phenotype after each of the 3 phases of expression: Grow, Rotate, and Zoom. The expression mechanism is described earlier in detail in 4.3.3, with the overall approach described in Algorithm 17. Also, the three phases are described in Algorithms 18, 22, and 28 respectively.

Examples in this chapter are generally presented in increasing complexity, beginning with simple and complex growth, followed by examples with simple and complex rotation which show the effect of different angle contiguity values.

Unless specifically indicated, all examples in this chapter were created with a target size
equal to the Ottawa segment (and consequently use the same minimum street length value). Only two examples use a city side length of half that of the Ottawa segment to illustrate the effect of city side length on expression. The city design based on Ottawa is discussed in detail in Chapter 7, and it is not necessary to be familiar with it at this point.

5.2 Genotype Layer Surface

Before exploring how different leaf values and angle contiguity values affect expression, it is important to understand what is meant by a genotype layer’s surface. Essentially, the surface is the result of stretching a layer’s leaf values over its surface, ignoring boundaries between identical nodes. For example, the genotype layers \( \{0\} \) and \( \{0,\{0,0,0,0\},0,0\} \) have identical surfaces. The surface becomes apparent when mapping leaf values to colour intensity (first introduced in 4.3.1).

This concept is important as it means that the structure of a tree does not necessarily affect its expression. Rather, the expression is based on its surface. The two genotypes previously mentioned would express to the same phenotype which is shown in Figure 5.2.

5.3 Simple Growth

The simplest form of expression includes growth with no rotation. In this section, examples of this simple growth are presented. Specifically, rotation layers are used whose leaf values are only zero.

5.3.1 All Zeros

During growth, Algorithm 18 is used to determine grid size as discussed in 4.3.3. When the surface of a given genotype layer is equal to zero, a maximum specified side length is returned, which is equal to the size of the whole phenotype (1). This is illustrated in expressing the genotype shown in Figure 5.1, whose corresponding phenotype is shown in Figure 5.2. This is the minimum ‘density’ of growth produced by the system.
As discussed in 4.3.3, the minimum street length (determined from the city side length) has an effect, except when the layer surface is equal to zero. This can be seen by expressing the genotype shown in Figure 5.1 again, this time using a city side length value equal to half of that used for the Ottawa segment. The resulting phenotype is shown in Figure 5.3, which, as expected, is identical to Figure 5.2. The next section will demonstrate this is not the case with non-zero surface values.
5.3.2 All Ones

The fullest grid density produced by the system during growth will depend on the minimum street length specified. The previous case, in which the underlying genotype layer contains only zeros, is the only case in which the minimum street length will not have any effect (for that layer).

The genotype used is shown in Figure 5.4. When expressed with a target city side length equal to that of the Ottawa segment, a resulting phenotype is produced, which is shown in Figure 5.5. Unlike with the all zeros example, the city size (and corresponding minimum street length) has an effect. Expressing the same genotype again with half the city side length of the Ottawa segment produces the phenotype seen in Figure 5.6.
Because the Ottawa city side length is larger, it yields a smaller minimum street length value than the Half Ottawa segment. Consequently, the resulting grid is more dense. This result is expected, and makes intuitive sense as the larger the area the result is meant to reflect, the more dense and ‘zoomed out’ one would expect the road network to appear. Values on the genotype layer surface will effectively scale between these two extremes.

### 5.4 Complex Growth

While a single density value might be sufficient to describe particular road networks, the road networks of real cities tend to be complicated and exhibit varying density. One advantage
of CityBreeder’s representation is that it allows for different values in different regions of a single design. Its quadtree-based structure allows for more ‘detail’ to be added in areas which exhibit greater local variations. The examples in this section show more complicated growths, with different densities in different regions of the city design. All examples in this section were produced with a target size equal to the Ottawa segment.

5.4.1 North West Density

In this example, a design is produced which is more dense in its north west region. The genotype is shown in Figure 5.7. This property is reflected in higher genotype values in leaves which map in the genotype’s surface to that particular area. This property can clearly be seen in the resulting phenotype (Figure 5.8).

\[
\begin{array}{ccc}
0.95 & 0.70 & 0.95 & 0.70 \\
0.70 & 0.70 & 0.70 & 0.70 \\
\end{array}
\]

(a) Scale X  (b) Scale Y  (c) Angle

Figure 5.7: North West Density Genotype

\[
\begin{array}{ccc}
\end{array}
\]

(a) Grow  (b) Rotate  (c) Zoom

Figure 5.8: North West Density Growth Expression
As can be seen, a somewhat organic transition occurs between regions of different density. It can also be observed that the system includes a certain degree of bias as the expression mechanism places its initial seed intersection in the top left corner. Therefore, it is more likely that the top left region will be more ordered as a regular grid than the other areas. The next example illustrates this further.

5.4.2 South East Density

This example is similar to the previous one, but the area of density is in the south east. Its genotype (Figure 5.9) expresses into the phenotype seen in Figure 5.10. The phenotype clearly demonstrates the differences in density with an organic transition between them. As with the previous example, some bias in order or grid regularity can be seen in the north west region. This example is included as it demonstrates that the result is not symmetrical with the north west density example—one could not simply rotate either final phenotype by 180° to get the same result. The placement of the initial seed does introduce some bias into the system.

![Figure 5.9: South East Density Genotype](image)

(a) Scale X  (b) Scale Y  (c) Angle

\[
\begin{array}{cc|cc}
0.70 & 0.70 & 0.70 & 0.70 \\
0.70 & 0.95 & 0.70 & 0.95 \\
\end{array}
\]

0.00
While it is more likely that the top left region will be more ordered as a regular grid than other regions, it is possible that grids can ‘re-align’ or ‘re-connect’ to produce tight grids again after diverging. This effect could be alleviated by mechanisms such as a randomized or parameterized initial starting location, multiple initial growth seeds, or additional ‘snapping’ to intersections during growth. The relative advantages and challenges of these potential avenues are discussed further in the Future Work in 9.2.

5.4.3 Central Density

The previous two examples show that variations can be introduced in different quadrants. It is relatively straightforward given the genotype’s quadtree-based implementation, where a root will have four subtrees (unless it is a leaf). However, the representation also allows variation to be introduced in other regions, but may require additional tree depth to do so.

In this example, a dense central region is produced using a more complex genotype. As a ‘central’ region cannot be described purely in terms of a design’s four quadrants, additional tree depth is required. The genotype in Figure 5.11 is more complex than in previous examples, and expresses into the phenotype shown in Figure 5.12.
Variations in grid density can be introduced into a phenotype at any position. The only limiting constraints are the maximum tree depth, and minimum street length (as derived from city side length).

5.5 Simple Rotation

In the previous section, all examples had rotation layers consisting solely of leaves with the value 0. In this section, different rotation values are illustrated. The same scale x and scale y layers are used with uniform rotation layer values of 0, 0.25, 0.5, 0.75, and 1.

The angle contiguity value does not affect the expression here, as all enclosures share the same proposed angle value. It only plays a role where there is variation in angle values. In the next section, the effect of this value is illustrated.
5.5.1 Simple Rotation 0

While the examples in the previous section did not discuss rotation, the rotation phase did operate, but had no effect given the values of zero present. The examples in this section have uniform scale x and scale y values of 0.95 and 0.75 respectively. As this particular grid density was not shown in the previous section, it is included here before modifying the angle values. The genotype in Figure 5.13 is expressed to yield the phenotype seen in Figure 5.14.

(a) Scale X  
(b) Scale Y  
(c) Angle

Figure 5.13: Simple Rotation 0 Genotype

(a) Grow  
(b) Rotate  
(c) Zoom

Figure 5.14: Simple Rotation 0 Expression

5.5.2 Simple Rotation 0.25

The angle layer is modified to 0.25, introducing a rotation of 1/4 of the angle between 0 and 90. In other words, the whole grid is rotated around its center of mass by 22.5°. The genotype in Figure 5.15 is expressed to yield the phenotype seen in Figure 5.16.
5.5.3 Simple Rotation 0.5

The genotype in Figure 5.17 is expressed to yield the phenotype seen in Figure 5.18, demonstrating a rotation of 45°.
5.5.4 Simple Rotation 0.75

The genotype in Figure 5.19 is expressed to yield the phenotype seen in Figure 5.20, demonstrating a rotation of $67.5^\circ$. 

```
0.95  0.75  0.75
(a) Scale X  (b) Scale Y  (c) Angle
```

Figure 5.19: Simple Rotation 0.75 Genotype
5.5.5 Simple Rotation 1

The rotation phase of the expression mechanism interprets leaf values as reflecting an angle between 0 and 90. If it mapped to higher values (such as 180 or 360) as it would essentially overwrite the distinction between scale x and scale y values. Rotations beyond 90° are possible, but require the scale x and y values to be reversed, which is possible through evolution.

The genotype in Figure 5.21 is expressed to yield the phenotype seen in Figure 5.22. As expected, it is identical to Figure 5.14, which is expressed with a genotype angle value of 0.

![Figure 5.20: Simple Rotation 0.75 Expression](image)

![Figure 5.21: Simple Rotation 1 Genotype](image)
5.6 Complex Rotation and Angle Contiguity Values

Whereas complex growth is dependent on the genotype’s scale layers and the minimum street length, complex rotation is dependent on the genotype’s angle layer and the angle contiguity value. This complex rotation arises when variations in angle values are present.

Examples in this section use a genotype whose angle layer suggests rotation in the north east quadrant. These examples illustrate the effect that different angle contiguity values have when expressing the same genotype.

In general, a smaller angle contiguity value will result in a larger number of separate contiguous enclosure groups, each with an angle value corresponding more closely to the values determined from the angle layer of the genotype. By contrast, a larger angle contiguity value will produce fewer groups with an averaged angle value.

The genotype used for each example in this section is shown in Figure 5.23.
It is worth observing that some of the rotation layers separate along the border of the north east quadrant. A close look at the result of Grow shows that the phenotype is not exactly divided by edges at $x = 0.5$ or $y = 0.5$ (neatly around its north east quadrant). This suggests that regions along this border will have proposed rotation values weighted from different parts of the genotype’s angle layer surface.

5.6.1 North East Rotated, Angle Contiguity 1

An angle contiguity value of 1 is equivalent to having no angle contiguity tolerance. Once the proposed angle value for each enclosure is grouped as described in Algorithm 27 in Chapter 4, adjacent enclosures are grouped only if they have exactly the same proposed angle value. The resulting phenotype is shown in Figure 5.24. An angle contiguity value of 1 will tend to produce phenotypes with a larger number of enclosure groups.
5.6.2 North East Rotated, Angle Contiguity 3

In this example, the *angle contiguity value* is slightly larger, which produces fewer enclosure groups. As can be seen in Figure 5.25, the enclosure closest to the middle of the phenotype is no longer separate as it was in 5.24.

![Figure 5.25: North East Rotated, Angle Contiguity 3 Expression](image)

5.6.3 North East Rotated, Angle Contiguity 5

An *angle contiguity value* of 5 produces a result which may intuitively reflect closest to what one might expect from the genotype, as can be seen in Figure 5.26. These expectations may vary depending on the context, which is why this value is provided as a parameter which can be adjusted.

![Figure 5.26: North East Rotated, Angle Contiguity 5 Expression](image)
5.6.4 North East Rotated, Angle Contiguity 50

Figure 5.27 shows the result of a large contiguity value, in this case finding one single enclosure group, and rotating according to a weighted average of the enclosures’ angle values.

![Figure 5.27: North East Rotated, Angle Contiguity 50 Expression](image)

(a) Grow  (b) Rotate  (c) Zoom

5.7 Summary

In this chapter, the genotype-to-phenotype expression mechanism was explored through examples which illustrate how a design can be produced with variations in grid density in the x and y dimensions, as well as differences in angle. These variations suggest that the representation is more expressive than simpler alternative representations which might evolve fewer properties.

The examples also illustrate that the system expresses genotypes into phenotypes in a predictable, intuitive manner. Furthermore, the genotypes and phenotypes examined together show that the representation chosen is more ‘transparent’ than some alternatives, as there exists a relative visual correspondence between the two.
Chapter 6

Simple Evolution

6.1 Introduction

Discovering the genes of existing city designs through evolution is a necessary step before being able to blend them. However, before examining this process on real city designs in the next chapter, the same operation is presented and discussed here, operating on several simpler contrived examples. It is useful to look at these cases as the genes of real cities may be difficult to evolve, and ‘good’ results may still include some error. In this chapter, it is shown that that the system is able to discover the genetic material of simple examples with perfect or very high accuracy, and illustrates some of the strengths and limitations of the representation. Additionally, the effects of individual operators are briefly explored.

It should be noted that fitness values in this chapter and the ones that follow are shown with 5 decimal places, but in the implementation are stored as doubles in Java [64].

6.2 Simple Targets

Simple evolution experiments are presented on three example designs. These example phenotypes were generated by expressing hand-coded genotypes. Each uses the same experimental parameters, with ‘reasonable values.’ The effect of each operator is examined later in the chapter. The experimental settings used in these experiments are shown in Table 6.1.
These values were selected as ‘reasonable’ for a number of reasons. In most of the GP literature examined, experiments were often conducted with a high crossover probability and low mutation probability. Early test experiments during the development of the system included only one mutation probability value and one for crossover and were set to 0.9 and 0.1 respectively. As discussed in Chapter 4, early test experiments had difficulty finding good results, as the mutation operators had been implemented in too powerful a fashion, and consequently mutations were not beneficial. Once the mutation operators were ‘weakened’ and more were introduced, they were all given this small probability of executing, and crossover was lowered slightly. This immediately produced better results.

The values for population size and number of generations were also selected after a number of ‘test’ experiments which suggested these values worked well in allowing the population to converge. Also, Rank Selection and Uniform Selection were both implemented, and used in that order for the two rounds of selection as one source suggests that Rank Selection has several excellent properties, such as “[reduc[ing] selection pressure under high fitness variance” and that it “maintains selection pressure under low fitness variance,” [63, p. 15]. It also recommended using Uniform Selection in one of the rounds of selection, or else the selection pressure would be too high [63, p. 9].

Table 6.1: Simple Evolution Experiment Settings

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Γ</td>
<td>Population Size</td>
<td>250</td>
</tr>
<tr>
<td>Δ</td>
<td>Number of Generations</td>
<td>50</td>
</tr>
<tr>
<td>Θ</td>
<td>Crossover Probability</td>
<td>0.7</td>
</tr>
<tr>
<td>Ω</td>
<td>Ratio of Basic to Weak-Context Preserving Crossover</td>
<td>0.5</td>
</tr>
<tr>
<td>Ξ</td>
<td>Expand Leaf to Quadtree Mutation Probability</td>
<td>0.1</td>
</tr>
<tr>
<td>Π</td>
<td>Compress Leaves-Only Quadtree to Leaf Mutation Probability</td>
<td>0.1</td>
</tr>
<tr>
<td>Σ</td>
<td>Gaussian Mutate Leaf Value Mutation Probability</td>
<td>0.2</td>
</tr>
<tr>
<td>Ψ</td>
<td>Permute Leaves-Only Quadtree Subtrees Probability</td>
<td>0.1</td>
</tr>
<tr>
<td>Φ</td>
<td>Permute Quadtree Subtrees Probability</td>
<td>0.1</td>
</tr>
<tr>
<td>Ψ</td>
<td>Angle Contiguity Window Value</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Selection Mechanisms</td>
<td>Rank, Uniform</td>
</tr>
<tr>
<td></td>
<td>Number of Runs</td>
<td>5</td>
</tr>
</tbody>
</table>
6.2.1 Experiment 1

The first simple target is a simple grid, with different scale x and y values, at a uniform angle. The target phenotype and genotype from which it was derived are shown in Figures 6.1 and 6.2.

![Figure 6.1: Simple Experiment 1 Source Genotype](image)

<table>
<thead>
<tr>
<th>Scale X</th>
<th>Scale Y</th>
<th>Angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.97</td>
<td>0.90</td>
<td>0.25</td>
</tr>
</tbody>
</table>

Figure 6.1: Simple Experiment 1 Source Genotype
The best and average fitnesses over 5 runs are shown in Figure 6.3. In all 5 runs, an ideal candidate with fitness 1.0 was found. In the 5 runs, the earliest a 1.0 result was found was at generation 25, and the latest in generation 38. The results show that the best and average fitness measures increase over time as expected.
The best result phenotypes from each of the 5 are visually similar. One of the ideal results is shown side-by-side with the original target in Figure 6.4. Rather than showing all 5 phenotypes created individually, a composite image of all 5 superimposed can be seen in Figure 6.5. This composite illustrates that the results are visually similar.
Figure 6.4: Target and Evolved Simple Experiment 1
Figure 6.5: Composite of Evolved Phenotypes for Simple Example 1
Though all 5 runs found an ideal solution, the genotypes and phenotypes found differ slightly from both the targets, and from each other. This results from the fact that the fitness evaluation method implicitly includes an accuracy measure, as difference maps are created at a specified resolution—\( \iota \), described in 4.5.1. The higher the resolution, the more accurate the fitness measure will be, but the longer it will take to calculate. All five phenotypes are considered perfect results, their minor differences within an amount made implicitly acceptable by the value selected for \( \iota \). The nature of this ‘implicitly acceptable amount’ is discussed in more detail in that earlier section. The accuracy is also influenced slightly by the design decision to round angle values to the nearest whole number, as shown in Algorithm 27. The genotypes of each are shown in Figures 6.6 to 6.10. Additionally, it should be recalled that minor offsets between intersections do not themselves affect the fitness rating as it is the properties of the enclosures which are considered.

![Figure 6.6: Simple Experiment 1 Run 1 Genotype](image)

![Figure 6.7: Simple Experiment 1 Run 2 Genotype](image)
These minor variations could be reduced if a larger difference map size is specified, but at a cost in run time. However, these results with high accuracy do intuitively visually resemble the target closely. It is important that results with high fitness also visually correspond as expected, otherwise, the fitness function could not be considered to be a good measure of its similarity.
6.2.2 Experiment 2

This example is similar to Experiment 1, but with a different angle. This target was initially selected for these experiments to show that the system could find angle values that were both in the range [0, 45) (Experiment 1) and [45, 90] (Experiment 2). This was prudent, as an early iteration of the system had difficulty distinguishing the two. The system no longer has difficulty with this. The target phenotype and genotype from which it was derived can be seen in Figures 6.11 and 6.12.

<table>
<thead>
<tr>
<th>0.90</th>
<th>0.97</th>
<th>0.75</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Scale X</td>
<td>(b) Scale Y</td>
<td>(c) Angle</td>
</tr>
</tbody>
</table>

Figure 6.11: Simple Experiment 2 Source Genotype
The best and average fitnesses over 5 runs are shown in Figure 6.13. As with the previous experiment, an ideal candidate with fitness 1.0 was found in each run. In the 5 runs, the earliest a 1.0 result was found was at generation 36, and the latest in generation 45. The results show that the best and average fitness measures increase over time as expected.
In similar fashion to the previous experiment, the best result phenotypes from each of the 5 runs appear very similar. One of the ideal results is shown side-by-side with the original target in Figure 6.14. Rather than showing all 5 phenotypes created individually, a composite image of all 5 superimposed can be seen in Figure 6.15. This composite illustrates that the results are very similar.
Figure 6.14: Target and Evolved Simple Experiment 2
Figure 6.15: Composite of Evolved Phenotypes for Simple Example 2
The genotypes of each best result are shown in Figures 6.16 to 6.20.

(a) Scale X
(b) Scale Y
(c) Angle

Figure 6.16: Simple Experiment 2 Run 1 Genotype

(a) Scale X
(b) Scale Y
(c) Angle

Figure 6.17: Simple Experiment 2 Run 2 Genotype

(a) Scale X
(b) Scale Y
(c) Angle

Figure 6.18: Simple Experiment 2 Run 3 Genotype
The evolved phenotype corresponds visually with the target phenotype, and its genotype can be seen to have similar values to the target’s genotype, reinforcing the conclusion that a good match has been found.

6.2.3 Experiment 3

This example explores a variation in angle, with the north east quadrant rotated. The target phenotype and genotype from which it was derived can be seen in Figures 6.21 and 6.22.
The best and average fitnesses over 5 runs are shown in Figure 6.23. Unlike the previous runs, an ideal solution with fitness 1.0 was not found. However, the best fitness in each run
was better than 0.98, and it is possible that an ideal result would have been found given a larger population size or number of generations.

![Figure 6.23: Evolution Fitness for Simple Example 3](image)

In each run, by generation 50, two of the runs had found their best result in generation 40, while three others had in generation 49. This suggests that given more generations or a larger population, an ideal solution might have been found. As there is slightly more variation among the five phenotypes found than was present in the previous two examples, each of the best phenotypes found are shown followed by their genotypes in Figures 6.24 to 6.33. Additionally, a composite can be seen in Figure 6.34.
Figure 6.24: Simple Experiment 3 Run 1 Phenotype. Fitness 0.98268

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.95</td>
<td>0.96</td>
<td>0.50</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.03</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.03</td>
</tr>
</tbody>
</table>

(a) Scale X  (b) Scale Y  (c) Angle

Figure 6.25: Simple Experiment 3 Run 1 Genotype
Figure 6.26: Simple Experiment 3 Run 2 Phenotype. Fitness 0.99214

Figure 6.27: Simple Experiment 3 Run 2 Genotype
Figure 6.28: Simple Experiment 3 Run 3 Phenotype. Fitness 0.98269

(a) Scale X

(b) Scale Y

(c) Angle

Figure 6.29: Simple Experiment 3 Run 3 Genotype
Figure 6.30: Simple Experiment 3 Run 4 Phenotype. Fitness 0.985560

Figure 6.31: Simple Experiment 3 Run 4 Genotype

<table>
<thead>
<tr>
<th></th>
<th>(a) Scale X</th>
<th>(b) Scale Y</th>
<th>(c) Angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.95</td>
<td>0.96</td>
<td>0.00</td>
<td>0.47</td>
</tr>
<tr>
<td></td>
<td>0.96</td>
<td>0.00</td>
<td>0.08</td>
</tr>
<tr>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Figure 6.32: Simple Experiment 3 Run 5 Phenotype. Fitness 0.98903

Figure 6.33: Simple Experiment 3 Run 5 Genotype

<table>
<thead>
<tr>
<th>(a) Scale X</th>
<th>(b) Scale Y</th>
<th>(c) Angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.96</td>
<td>0.96</td>
<td>0.02 0.49</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.02 0.00</td>
</tr>
</tbody>
</table>

(a) Scale X    (b) Scale Y    (c) Angle
Most of these results correspond closely to the target phenotype; in particular, the last two runs. In Figure 6.35, the result of run 4 is shown side-by-side with the target.

In this particular case, the results of runs 4 and 5 appear more similar than the higher fitness result found in run 2. However, given more generations or a larger population, it is likely that an ideal solution would be found. Additionally, a larger difference map size would likely penalize the result from run 3 whose additional small enclosures might be missed or averaged during the creation of a smaller difference map.
6.3 Individual Operators

In this section, experiments are presented which attempt evolve the same target as in the preceding Simple Experiment 3, using each genetic operator individually. These are presented to offer some (limited) understanding of the relative power of each. The experiments in this section were run with the same parameter settings as those in the preceding section, the only difference being the probabilities for each operator. Specifically, for each operator, experiments were run with two probability values: 0.1 and 0.9. This is equivalent to testing each operator in isolation on ‘low’ and ‘high.’ While the best and average fitness values in some of these experiments plateau, others do not. These results do not exhaustively examine the effects of each operator, but do speak to how well they perform in a given number of generations (50). Future work might include experiments which operate for a larger number of generations.

A thorough examination of effects of different parameter settings (and combinations of operator probabilities) would be a large undertaking, and is beyond the scope of this thesis. However, the results presented in this section provide a few data points for such a parameter analysis and allow some conclusions to be drawn regarding their relative utility.
It is useful to note that the error is relatively larger on the best fitness values in these examples than in the previous section, because the ‘best’ results which are found will vary considerably between runs if individual operators are unable to climb out of local maxima.

In each section, the phenotype and genotype for the best result is shown. Because some of the genotypes have large trees which are hard to read, colour visualizations are also provided.

6.3.1 Basic Crossover

Basic Crossover, discussed in detail in 2.3.2 and 4.4.1, creates two child genotypes by copying two parents and swapping a pair of randomly selected subtrees. The resulting best and average fitness values can be seen in Figure 6.36.

![Figure 6.36: Basic Crossover Fitness](image)

The fittest solution evolved is shown in Figures 6.37 to 6.39. It has a fitness value of 0.92773.
Figure 6.37: Basic Crossover Only Evolved Phenotype

Figure 6.38: Basic Crossover Only Evolved Genotype
The results do not appear to closely correspond to the target, but certain corresponding features do appear, such as the angle, and scale x density. This is confirmed by examining the genotype which suggests that the scale y layer contains values which differ most from the target’s genotype.

6.3.2 Weak Context-Preserving Crossover

Weak Context-Preserving Crossover, discussed in detail in 2.3.2 and 4.4.1, creates two child genotypes in a manner similar to Basic Crossover, except that the subtrees chosen for swapping must be located along the same path from their respective roots. This mechanism will transfer information in a less random manner than Basic Crossover, since the tree data will end up in approximately the same spatial location. The resulting best and average fitness values can be seen in Figure 6.40.
In this experiment, Weak Context-Preserving Crossover can be seen to converge towards a good value considerably more quickly than Basic Crossover.

The fittest solution evolved is shown in Figures 6.41 to 6.43. It has a fitness value of 0.98464.
Figure 6.41: Weak Context-Preserving Crossover Only Evolved Phenotype

Figure 6.42: Weak Context-Preserving Crossover Only Evolved Genotype
This result corresponds much more closely with the target phenotype than the result found through Basic Crossover. This is also supported by its higher fitness value. The few observable differences lie along the boundary between the regions with different proposed angle values. This is the same area which appeared susceptible to the same issue earlier in 6.2.3. An examination of the genotype shows that the correct scale x and scale y values were found, but that the angle is inaccurate.

The results found by the two crossover mechanisms are not perfect, but bare a resemblance to the target. As will be shown, the mutation operators are not capable of this alone.

### 6.3.3 Expand Leaf to Quadtree Mutation

The first mutation operator, Expand Leaf to Quadtree Mutation, is discussed in detail in 4.4.2. It has the effect of creating deeper trees within the population. In the absence of other operators, given enough generations, this will produce a population where all individuals have trees full to the maximum specified depth. The resulting best and average fitness values can be seen in Figure 6.44.
This operator has no utility in isolation because the genotypes before and after this genetic expansion will express into identical phenotypes. However, this operator creates depth, which can be utilized by other operators to positive effect.

The fittest solution evolved is shown in Figures 6.45 to 6.47. It has a fitness value of 0.63320.
Figure 6.45: Expand Leaf to Quadtree Mutation Only Evolved Phenotype

Figure 6.46: Expand Leaf to Quadtree Mutation Only Evolved Genotype
This result bares little resemblance to the target, as one would expect from the low fitness rating.

6.3.4 Compress Leaves-Only Quadtree to Leaf Mutation

Compress Leaves-Only Quadtree to Leaf Mutation, discussed in 4.4.2, has the opposite effect from the previous operator, as it creates shallower trees within the population. In the absence of other operators, given enough generations, this will produce a population where all individuals have a single leaf node on each genotype layer. The resulting best and average fitness values can be seen in Figure 6.48.
As with the previous operator, this one does little on its own to improve fitness. However, it can serve to limit the average tree size, since most other genetic operators tend to increase tree length. This can be advantageous in cases where simpler genotypes are advantageous but other genetic operators have little probability of producing such a state.

The fittest solution evolved is shown in Figures 6.49 to 6.51. It has a fitness value of 0.64862.
Figure 6.49: Compress Leaves-Only Quadtree to Leaf Mutation Only Evolved Phenotype

Figure 6.50: Compress Leaves-Only Quadtree to Leaf Mutation Only Evolved Genotype
It may be unexpected to see such a deep genotype when the population’s average tree depth is decreasing over time. This result is possible because the system uses elitism, keeping the best individual from either round of selection, carrying it through to the next generation. In this case, it appears that an elite from an early generation was carried through to the end of the evolution given its high fitness relative to the rest of the population, and that the simpler genotypes created through this mutation alone did express a fitter result.

### 6.3.5 Gaussian Mutate Leaf Value Mutation

Gaussian Mutate Leaf Value Mutation, discussed in 4.4.2, does not affect tree depth, but will change a leaf value within a genotype. This operator appears to be more powerful in isolation than either of the previous two mutation operators, but far less so than the crossover operators. However, it is possible that its full potential may only be realized when it is appropriately coupled with other operators which modify a tree’s structure but cannot influence its leaf values. In this regard, it might be thought of as more akin a ‘local search’ unlike the more powerful ‘global search’ of crossover. The resulting best and average fitness values can be seen in Figure 6.52.
The fittest solution evolved is shown in Figures 6.53 to 6.55. It has a fitness value of 0.72598.
Figure 6.53: Gaussian Mutate Leaf Value Mutation Only Evolved Phenotype

Figure 6.54: Gaussian Mutate Leaf Value Mutation Only Evolved Genotype
6.3.6 *Permute Leaves-Only Quadtree Subtrees Mutation*

Permute Leaves-Only Quadtree Subtrees Mutation, discussed in 4.4.2, does affect a tree’s structure, but in a less destructive (or constructive) manner than the crossover operators. The variant which operates on leaves-only subtrees can also be thought of as not affecting the tree’s structure, as it merely swaps leaf values, and will not change subtree depth at any point within the tree. The resulting best and average fitness values can be seen in Figure 6.56.

Given a sufficiently deep tree within the population, it is conceivable that this operator will find a good result if given a sufficiently (and unreasonably) large number of generations during which to operate.
The fittest solution evolved is shown in Figures 6.57 to 6.59. It has a fitness value of 0.72546.
Figure 6.57: Permute Leaves-Only Quadtree Subtrees Mutation Only Evolved Phenotype

Figure 6.58: Permute Leaves-Only Quadtree Subtrees Mutation Only Evolved Genotype
6.3.7 Permute Quadtree Subtrees Mutation

Similar to the previous operator, Permute Quadtree Subtrees Mutation (discussed in 4.4.2) swaps the subtrees of a given quadtree. However, it does not operate exclusively on quadtrees with all leaf values. It is slightly more powerful than the previous operator as it modifies tree structure, changing the maximum depth at different points within the tree, though it cannot change the maximum tree depth. The resulting best and average fitness values can be seen in Figure 6.60.
These results show that this operator is more useful than its leaves-only sibling on this example. However, using too many powerful mechanisms may be too destructive, yielding fewer useful mutations once local optima have been reached. For this reason, ‘weaker’ operators are employed alongside more powerful ones in the rest of the experiments in this thesis.

The fittest solution evolved is shown in Figures 6.61 to 6.63. It has a fitness value of 0.76167.
Figure 6.61: Permute Quadtree Subtrees Mutation Only Evolved Phenotype

Figure 6.62: Permute Quadtree Subtrees Mutation Only Evolved Genotype
Each of the phenotypes created by mutation alone has low fitness and bears little resemblance to the target. That the result in Simple Experiment 3 (6.2.3) reaches a better solution than crossover alone suggests these mutation operators do help in pushing the evolution towards fitter solutions.

This section examined the abilities of specific individual operators during a limited number of generations to evolve a simple example, and showed that some are more capable than others. However, it is likely that a more effective setup employs several of these to some degree. A more complete examination of the parameter space would be appropriate future work. The settings chosen for other experiments in this thesis make use of each of these operators in small measure, in order to bring a variety of tools to the evolutionary process, to try to increase diversity and give it the best chance to find good solutions.

The performance of each operator in the previous experiments is summarized in Table 6.2 which shows them ranked according to the best fitness each could find.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Operator</th>
<th>Best Fitness Achieved</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Weak Context-Preserving Crossover</td>
<td>0.98464</td>
</tr>
<tr>
<td>2</td>
<td>Basic Crossover</td>
<td>0.92773</td>
</tr>
<tr>
<td>3</td>
<td>Permute Quadtree Subtrees Mutation</td>
<td>0.76167</td>
</tr>
<tr>
<td>4</td>
<td>Gaussian Mutate Leaf Value Mutation</td>
<td>0.72598</td>
</tr>
<tr>
<td>5</td>
<td>Permute Leaves-Only Quadtree Subtrees Mutation</td>
<td>0.72546</td>
</tr>
<tr>
<td>6</td>
<td>Compress Leaves-Only Quadtree to Leaf Mutation</td>
<td>0.64862</td>
</tr>
<tr>
<td>7</td>
<td>Expand Leaf to Quadtree Mutation</td>
<td>0.63320</td>
</tr>
</tbody>
</table>

Table 6.2: Summary of Individual Operators Simple Evolution Results
6.4 Summary

This chapter examined *CityBreeder’s* ability to discover the genes of simple contrived examples. In these examples it performed well, finding results which were optimal or at the least had very high fitness. In these cases, the result also corresponded visually with the target phenotype as expected. The results demonstrate clearly that the representation, genetic operators, and fitness function can be used to evolve complex, overlapping grids such as found in city road network layouts.

A brief examination of the capabilities of specific genetic operators given a limited number of generations was also presented in this chapter, suggesting that some have more power than others. However, the best settings might very well include a combination of these operators. A thorough investigation of the parameter space is a large task and is appropriate future work.

Having examined *CityBreeder’s* ability to discover the genes of simple examples, it is now appropriate to examine that capability applied to real city data. This will be examined in the next chapter.
Chapter 7

Discovering City Genes

7.1 Introduction

Before city designs can be bred together to create new ones, their genes must first be discovered. The previous chapter examined this discovery process on simple contrived examples, finding high fitness solutions which correspond visually with their targets. In this chapter, this technique is demonstrated on several city designs derived from real city data. These city designs are derived from segments or neighbourhoods of Ottawa, New York, and Paris. For each city, information regarding its import is offered before presenting the results of discovering its genes.

7.2 The Import Process

The process of importing OpenStreetMap data is described in detail in 4.2.1. However, it should be noted that data that is used in this thesis was exported from Osmosis using a planetary data file from OpenStreetMap. This was done to avoid the limitation placed on the number of nodes exported through OpenStreetMap’s web interface. These tools are discussed earlier in 2.4.2. The plantary data file was obtained from the OpenStreetMap website in August 2012. This is mentioned because screenshots of OpenStreetMap appear in Figures 7.3, 7.4, 7.15, 7.16, 7.26, and 7.27, which were taken in August 2013. As the data may have changed somewhat over the course of a year, these images are intended solely to give a sense of the placement and appearance of boundaries on the full map. Additionally, OpenStreetMap may round the input dimensions when placing the bounding box for use in the images, unlike Osmosis from which the OSM files were exported.

182
7.3 Ottawa

Ottawa is the capital of Canada, and home to Carleton University. The segment imported contains portions of Centretown, the Glebe and Dow’s Lake neighbourhoods.

7.3.1 Import

The information presented in this section summarizes the details of importing this particular segment. As discussed in 4.2.1, the system allows the user to specify the desired usable area from OSM data of any bounds. For the three imports in this chapter, larger areas were exported than required to ensure that enclosures overlapping the usable area’s boundary would not be clipped. The usable area and exported area coordinates are summarized in Tables 7.1 and 7.2.

<table>
<thead>
<tr>
<th>Coordinate</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum longitude</td>
<td>−75.71086</td>
</tr>
<tr>
<td>Maximum longitude</td>
<td>−75.67735</td>
</tr>
<tr>
<td>Minimum latitude</td>
<td>45.39321</td>
</tr>
<tr>
<td>Maximum latitude</td>
<td>45.41623</td>
</tr>
</tbody>
</table>

Table 7.1: Ottawa Useable Coordinates

<table>
<thead>
<tr>
<th>Coordinate</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum longitude</td>
<td>−75.74200</td>
</tr>
<tr>
<td>Maximum longitude</td>
<td>−75.63400</td>
</tr>
<tr>
<td>Minimum latitude</td>
<td>45.36950</td>
</tr>
<tr>
<td>Maximum latitude</td>
<td>45.43650</td>
</tr>
</tbody>
</table>

Table 7.2: Ottawa Exported Coordinates

To give an indication of where these areas lie within the city, screenshots from OpenStreetMap are shown with these areas selected in Figures 7.1 and 7.2.
Figure 7.1: OpenStreetMap Screenshot for Ottawa with Useable Area Outlined

Figure 7.2: OpenStreetMap Screenshot for Ottawa with Exported Area Outlined
The results from importing the OSM file into *CityBreeder* is shown in Figure 7.3, and can be seen zoomed out in Figure 7.4 with the useable area demarcated by a green box in the center.

Figure 7.3: Ottawa Imported (Target) Phenotype
To later determine a candidate phenotype’s fitness in relation to this target city design, categorical maps are created for the imported design. This process is discussed earlier in 4.5. The three resulting categorical maps are displayed in Figure 7.5 with and without the enclosures’ oriented bounding boxes visible.
Figure 7.5: Ottawa Categorical Maps
A visualization with the added bounding boxes is useful as multiple phenotypes may map to the the same categorical map. The simplest such case would be two uniform grids with the same enclosure scales and angle, but where one is offset, such that the nodes and edges do not line up. More complicated cases might also arise. This quality of the fitness measure is discussed earlier in detail in 4.5.

### 7.3.2 Discovering Genes

Experiments similar to those presented in the preceding chapter were run to discover the genes for the Ottawa segment. The experimental parameters used are shown in Table 7.3. They are identical to the settings used in the previous chapter in 6.2, except that a larger number of generations are used (100).

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Γ</td>
<td>Population Size</td>
<td>250</td>
</tr>
<tr>
<td>Δ</td>
<td>Number of Generations</td>
<td>100</td>
</tr>
<tr>
<td>Θ</td>
<td>Crossover Probability</td>
<td>0.7</td>
</tr>
<tr>
<td>Ω</td>
<td>Ratio of Basic to Weak-Context Preserving Crossover</td>
<td>0.5</td>
</tr>
<tr>
<td>Ξ</td>
<td>Expand Leaf to Quadtree Mutation Probability</td>
<td>0.1</td>
</tr>
<tr>
<td>Π</td>
<td>Compress Leaves-Only Quadtree to Leaf Mutation Probability</td>
<td>0.1</td>
</tr>
<tr>
<td>Σ</td>
<td>Gaussian Mutate Leaf Value Mutation Probability</td>
<td>0.2</td>
</tr>
<tr>
<td>Υ</td>
<td>Permute Leaves-Only Quadtree Subtrees Probability</td>
<td>0.1</td>
</tr>
<tr>
<td>Φ</td>
<td>Permute Quadtree Subtrees Probability</td>
<td>0.1</td>
</tr>
<tr>
<td>Ψ</td>
<td>Angle Contiguity Window Value</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Selection Mechanisms</td>
<td>Rank, Uniform</td>
</tr>
<tr>
<td></td>
<td>Number of Runs</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 7.3: Discovering City Genes Ottawa Experiment Settings

The experimental results are shown in Figure 7.6.
The results indicate an increase in best and average fitness over time, plateauing at approximately generations 20 and 40 respectively. These runs found individuals with fitnesses of approximately 0.94.

The fittest individual was found in run 1, and had a fitness of 0.93985. It is shown in Figures 7.7 and 7.8. Given the genotype tree depth, an RGB visualization is included for quick visual reference.
This phenotype can also be seen side-by-side with the target phenotype in Figure 7.9, with the target’s non-areas (in this case, the canal and lake) superimposed on both images. It is useful to view it in this manner as the fitness function will not consider any portion of the candidate map in these regions when assessing fitness.

The import mechanism is not quite perfect, as it incorrectly identifies part of the canal in the north west as an enclosure instead of a non-area. This limitation is discussed earlier in 4.5.

From a visual examination, several observations can be made: the angle appears similar, and the enclosures are approximately the same shape as those in the target. It is also a good sign that larger enclosures were generated in the south west corner and in the south east by the canal’s ‘elbow.’ This corresponds with the target. It also appears that a solution
was found which rotated away a portion to create the larger, oddly-shaped region near the canal’s elbow. This is interesting as it demonstrates that the system can use rotation to create non-rectangular enclosures.

Though the enclosures appear to be generally the appropriate shape and angle, the evolved enclosures do generally appear slightly larger. An examination of the genotype reveals that the scale layers ‘hit’ the minimum size limit in many places by evolving 1.00 leaf values. It is probable that if this limit were changed that the evolutionary process might find even better results with slightly smaller and more accurate enclosures.

More information can be gathered regarding the fitness of this evolved individual by examining the difference maps which are used to determine similarity with the target, and consequently its fitness. These difference maps are shown in Figure 7.11. They were created from the target’s categorical maps, which are displayed in Figure 7.10 with and without the
Figure 7.9: Target and Evolved Ottawa

enclosures’ oriented bounding boxes visible.
Figure 7.10: Evolved Ottawa Categorical Maps
The difference maps appear grainy given the 100 pixel side length selected. The choice of resolution is discussed earlier in 4.5.1. From the difference maps, one may observe that each layer is largely dark, indicating high similarity, except in a few specific locations. The similarity calculated for each layer is summarized in Table 7.4. As discussed in 6.1, values are listed to 5 decimal places.

<table>
<thead>
<tr>
<th>Similarity Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale X similarity</td>
<td>0.94166</td>
</tr>
<tr>
<td>Scale Y similarity</td>
<td>0.93911</td>
</tr>
<tr>
<td>Angle similarity</td>
<td>0.93876</td>
</tr>
<tr>
<td>Weighted similarity (fitness)</td>
<td>0.93985</td>
</tr>
</tbody>
</table>

Table 7.4: Ottawa Similarity Values

The other runs all found best individuals with similar genotypes and phenotypes to the one just presented. Some of these other results are interesting for the fact that they show the system is able to produce results with enclosures smaller than the minimum specified amount by exploiting the expression process to create ‘offsets’ within the grid due to variations in scale layer values. The result of run 3 demonstrates this effect most clearly and is included below in Figure 7.12 (fitness 0.93876). However, despite this interesting effect, they fail to attain better fitness than the individual just presented.
All of these results share similar properties. The enclosures are of approximately the same shape as those in the target, and appear to be at the same angle. They also have slightly larger enclosures in the south west and near the canal’s elbow, also similar to the target. For this reason, they can be seen to intuitively, visually correspond. Increasing the minimum street length would likely achieve even better results.

7.4 New York

New York City, particularly Manhattan, is known for its grid-like street layouts. Consequently, one would expect the system to perform especially well on a segment from this city. The segment selected includes the area around Times Square.
7.4.1 Import

The useable area and exported area coordinates are summarized in Tables 7.5 and 7.6.

<table>
<thead>
<tr>
<th>Coordinate</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum longitude</td>
<td>−73.99966</td>
</tr>
<tr>
<td>Maximum longitude</td>
<td>−73.96829</td>
</tr>
<tr>
<td>Minimum latitude</td>
<td>40.74511</td>
</tr>
<tr>
<td>Maximum latitude</td>
<td>40.76666</td>
</tr>
</tbody>
</table>

Table 7.5: New York Useable Coordinates

<table>
<thead>
<tr>
<th>Coordinate</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum longitude</td>
<td>−74.03100</td>
</tr>
<tr>
<td>Maximum longitude</td>
<td>−73.92600</td>
</tr>
<tr>
<td>Minimum latitude</td>
<td>40.69560</td>
</tr>
<tr>
<td>Maximum latitude</td>
<td>40.81930</td>
</tr>
</tbody>
</table>

Table 7.6: New York Exported Coordinates

These areas can be seen highlighted in screenshots from *OpenStreetMap* in Figures 7.13 and 7.14. As noted in the Ottawa section, the data in the screenshot is more recent than that used for import.
Figure 7.13: *OpenStreetMap* Screenshot for New York with Useable Area Outlined

Figure 7.14: *OpenStreetMap* Screenshot for New York with Exported Area Outlined
The results from importing the OSM file into *CityBreeder* is shown in Figure 7.15, and is shown zoomed out in Figure 7.16 with the useable area demarcated by a green box in the center. The result of the import process is that the ‘major’ streets have been imported, roughly corresponding to those in yellow in Figures 7.13 and 7.14. The remaining streets could be imported by adding to the list of recognized way types in the import process (specified in 4.2.1). However, this segment is particularly useful in this form as it displays more variety in grid density. In addition to being derived from real city data, this will make it a better test of the system.

The categorical maps for the imported design are displayed in Figure 7.17 with and without the enclosures’ oriented bounding boxes visible.
Figure 7.15: New York Imported (Target) Phenotype
Figure 7.16: New York Imported (Target) Phenotype 1/7 Zoom
Figure 7.17: New York Categorical Maps
The imported phenotype has a number of interesting yet subtle variations in its features. It has a fairly uniform angle, with minor differences between the north and south. The scale x map shows a visible difference between the east and west, while the scale y map shows a visible difference between north and south.

### 7.4.2 Discovering Genes

Experiments were run to discover the genes for this target, using the same experimental parameters as those used for the Ottawa segment. These settings are listed earlier in Table 7.3. The experimental results are shown in Figure 7.18.

![Figure 7.18: Evolution Fitness for New York](image)

The results indicate an increase in best and average fitness over time, plateauing at approximately generations 30 and 40 respectively. These runs found individuals with fitnesses of approximately 0.975.

The fittest individual was found in run 4, and had a fitness of 0.97708. It is shown in Figures 7.19 and 7.20.
This phenotype can also be seen side-by-side with the target phenotype in Figure 7.21, with the target’s non-areas superimposed. Though there are not bodies of water in the
imported design, there are some small non-areas designated during the import process, which affects the fitness function. Further work to refine the import process or the addition of manual pre-processing of the data would eliminate this in future work.

From a visual examination, several observations can be made. The angle appears to correspond well, as do the enclosure scales except for the few narrow enclosures in the target’s south west quadrant. However, the evolved phenotype does include the subtle differences in scale between the east and west, and between north and south. This suggests a good result has been found.

Similar to the Ottawa result, though the enclosures appear to be approximately the same size as those in the target, they do appear slightly larger. An examination of the genotype reveals that, at least in the north on the scale y layer, the minimum street size limit is reached as 1.00 leaf values were evolved. Again, it is probable that if this limit were changed that the evolutionary process might find even better results with slightly smaller and more accurate enclosures. The categorical maps for the evolved individual are displayed in Figure 7.17 with and without the enclosures’ oriented bounding boxes visible. The difference maps are shown in Figure 7.23.

(a) New York Target Phenotype  (b) New York Evolved Phenotype

Figure 7.21: Target and Evolved New York
Figure 7.22: Evolved New York Categorical Maps
These difference maps appear almost uniformly dark, suggesting good similarity throughout the map. The narrow roads in the south which were not evolved do show up slightly brighter in the scale y map, standing out as one of the evolved individual’s weaknesses. The similarity values calculated are summarized in Table 7.7, revealing that the scale y map was noticeably lower in similarity.

<table>
<thead>
<tr>
<th>Similarity Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale X similarity</td>
<td>0.98507</td>
</tr>
<tr>
<td>Scale Y similarity</td>
<td>0.96556</td>
</tr>
<tr>
<td>Angle similarity</td>
<td>0.98068</td>
</tr>
<tr>
<td>Weighted similarity (fitness)</td>
<td>0.97708</td>
</tr>
</tbody>
</table>

Table 7.7: New York Similarity Values

The best individuals found in the other runs share similar properties with the best individual, but are slightly less convincing from an intuitive visual perspective. This is expected given their slightly lower fitness ratings.

7.5 Paris

Paris is a city with a complex road layout, exhibiting radial patterns, with roads emanating from central points. Given this system’s focus on the grid structure, it is anticipated that the system will not perform as well as it did with Ottawa and New York.
7.5.1 Import

The segment of Paris used includes the area around the Eiffel Tower, the École Militaire and extending south, including portions of Grenelle and Vaugirard. The useable area and exported area coordinates are summarized in Tables 7.8 and 7.9.

<table>
<thead>
<tr>
<th>Coordinate</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum longitude</td>
<td>2.27620</td>
</tr>
<tr>
<td>Maximum longitude</td>
<td>2.32550</td>
</tr>
<tr>
<td>Minimum latitude</td>
<td>48.83500</td>
</tr>
<tr>
<td>Maximum latitude</td>
<td>48.86710</td>
</tr>
</tbody>
</table>

Table 7.8: Paris Useable Coordinates

<table>
<thead>
<tr>
<th>Coordinate</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum longitude</td>
<td>2.25530</td>
</tr>
<tr>
<td>Maximum longitude</td>
<td>2.35730</td>
</tr>
<tr>
<td>Minimum latitude</td>
<td>48.81860</td>
</tr>
<tr>
<td>Maximum latitude</td>
<td>48.87590</td>
</tr>
</tbody>
</table>

Table 7.9: Paris Exported Coordinates

These areas can be seen highlighted in screenshots from OpenStreetMap in Figures 7.24 and 7.25. As previously noted, the data in the screenshot is more recent than that used for import. Also, as with Ottawa, the import is imperfect, with a small portion of water that is not identified as a non-area near the top of the map.
Figure 7.24: *OpenStreetMap* Screenshot for Paris with Useable Area Outlined

Figure 7.25: *OpenStreetMap* Screenshot for Paris with Exported Area Outlined
The result of importing the OSM file into CityBreeder is shown in Figure 7.26, and can be seen zoomed out in Figure 7.27 with the useable area demarcated by a green box in the center.

Figure 7.26: Paris Imported (Target) Phenotype
The categorical maps for the imported design are displayed in Figure 7.28 with and without the enclosures’ oriented bounding boxes visible.
Figure 7.28: Paris Categorical Maps
The vivid variations in intensity in the angle map hints at the difficulty the system may have with this target.

7.5.2 Discovering Genes

Experiments were run to discover the genes for this target, using the same experimental parameters used with the previous two city segments, except for two differences. Experiments were only run to generation 50, given how quickly the other experiments converged. Also, the experiments were run with two different angle contiguity values: 10 (same as previous experiments), and 1.

The experimental results are shown in Figures 7.29 and 7.30.

Figure 7.29: Evolution Fitness for Paris with Angle Contiguity 10
Both sets of runs indicate similar rates of improvement of best and average fitness over time, though those with angle contiguity 10 converged slightly faster. The best individuals found in all runs of both sets of experiments had fitness values of approximately 0.911. The fittest individual from each experiment set is shown in Figures 7.31 to 7.34.
Figure 7.31: Paris Angle Contiguity 10 Run 5 Phenotype

(a) Scale X

(b) Scale Y

(c) Angle

Figure 7.32: Paris Angle Contiguity 10 Run 5 Genotype
Figure 7.33: Paris Angle Contiguity 1 Run 2 Phenotype

<table>
<thead>
<tr>
<th></th>
<th>(a) Scale X</th>
<th>(b) Scale Y</th>
<th>(c) Angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.98</td>
<td>0.98</td>
<td>0.98</td>
<td>0.49</td>
</tr>
<tr>
<td>0.98</td>
<td>0.98</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 7.34: Paris Angle Contiguity 1 Run 2 Genotype

These two results are almost identical, with the second one having a slightly higher fitness of 0.91172. It is shown side-by-side the target phenotype in Figure 7.35 with the target's
non-areas superimposed.

This result is not as convincing as those evolved for Ottawa and New York. One might expect more diversity in angle, and smaller enclosures. Regarding angles, it is likely that the system is finding an average value, and in that regard operating as one would expect: to minimize error. However, an average will not be as accurate as a more varied and specific result. This result might seem somewhat ‘reasonable’ compared to some of the highly disordered potential solutions which exist in the solution space (such as some of the results encountered when testing genetic operators in isolation in Chapter 6).

More information can be gathered by examining the categorical maps, difference maps, and similarity values calculated from the difference maps, which are shown in Figure 7.36, Figure 7.37, and Table 7.10 respectively.
Figure 7.36: Evolved Paris Categorical Maps
These results suggest that the angle negatively impacted the fitness considerably more than the enclosure scales. Anecdotally, during experimentation (including a number of smaller runs during the development of CityBreeder), it seemed as though results exhibiting fitness values less than 0.93 did not appear to correspond visually. This observation is also consistent with the results presented in this chapter.

Reducing the minimum street length might help to yield better results. However, it does appear that the system is better suited to more grid-like designs. Methods of improving the system to better evolve other structures, such as radial designs will be discussed in the Future Work in 9.2.

### 7.6 Summary

In this chapter, the first phase of CityBreeder’s pipeline was tested with real city designs to discover their genes in advance of breeding the designs together. The experimental results suggest the system works well, achieving a broad visual correspondence with supporting
quantitative similarity values for fairly regular, grid-like city designs based on the segments from Ottawa and New York. The system had more difficulty achieving the same result for Paris, given its radial and irregular structures, but did manage to find ‘reasonable’ average values given its limitations. Methods of improving its performance with designs such as Paris will be discussed in the Future Work, while the next chapter examines the creation of new city designs using these evolved genes for Ottawa and New York.
Chapter 8

Breeding Cities

8.1 Introduction

In this chapter, the second phase of CityBreeder’s pipeline—Breeding Cities—is examined. Specifically, two demonstrations are presented which show its ability to create new designs from existing ones. This is possible once the genes of existing city designs have been discovered, as was done in the preceding chapter. This capability, combined with the ability to discover a city’s genes, enables the system to answer the problem statement.

While these demonstrations show that this capability has been achieved, they only explore this phase in a limited manner. As a result, specific examples are also presented showing genetic recombinations of real city designs in greater detail.

Although the software implementation is not a major focus in this thesis (as discussed earlier in the scoping section, 1.3), it is necessary to show part of the software in this chapter as part of the demonstrations. The design of the interface and specifics concerning its operation are discussed earlier in 4.4.3.

8.1.1 Speed

The experiments presented in this chapter and the preceding two examine the results found through the software but do not consider runtime. In Chapter 1, it was stated that the software would not be a focus of this thesis. However, one of the many requirements in the problem statement is that the system be fast. While scientific results concerning system speed are not presented, it was found that the Breeding Cities phase operated very quickly with the user waiting between 10 and 30 seconds between feedback windows (this includes evolutionary operations as well as graphical software processing). By contrast, experiments for Discovering City Genes tended to take several hours, given larger population sizes and numbers of generations.

These times are acceptable as the Discovery of City genes phase can be thought of as a
preparatory step that needs to be executed once. If this software was distributed to users, it could have genotype and phenotype representations for multiple real cities included. This would mean that the city designer would only have to directly engage with phase 2, which meets the ‘fast’ criteria established in Chapter 1.

8.2 Demonstrations

In creating a new city design using this system, the user may already have a preconceived idea concerning the properties of their intended final design, or alternatively may see ‘what looks good’ from generation to generation, implicitly possessing a shifting bias.

These two demonstrations illustrate how the system appears to the user, and shows that the system is capable of breeding specific traits. In the first demonstration, individuals are selected whose road networks appear most dense, while the second experiment favours those with roads appearing most sparse. Designs are selected as being ‘dense’ or ‘sparse’ based on a subjective visual analysis. Consequently, these two demonstrations are largely qualitative in nature, though analysis is performed both on qualitative aspects and through an examination of the numerical properties of specific genotypes.

A number of other potential experiments could be run, such as using other kinds of targets (evaluating rotation as well), gathering qualitative user impressions, and calculating objective fitness measures based on similarity with pre-selected targets. These other potential experiments are discussed more in the Future Work in 9.2.

Both experiments use the imported Ottawa design’s city side length as the target size, and are shown for 5 generations.

8.2.1 Dense Target

In this experiment, individuals are selected which appear most dense. Figures 8.1, 8.3, 8.4, 8.5, and 8.6 show the windows displayed to the user at each generation. This feedback occurs during preparation of round one of selection. Each screenshot was taken after the desired individuals had already been selected. These selected individuals are identified by a blue border which is toggled by the user clicking on the individual.

The first generation shows a relatively diverse population, despite several individuals appearing very similar to the New York design. The population exhibits a range of grid
densities between the Ottawa and New York scales, and individuals are also seen with angles approximately equivalent to those of the two initial designs. Consequently, the blending of the properties of the source designs is apparent. For example, the selected individual (second to the far right) appears to have approximately the grid density of New York at Ottawa’s angle, with perhaps additional mutation in one of the quadrants.

This can be confirmed by examining that individual’s genotype. That individual is shown in more detail in Figure 8.2. Comparing it with the genotypes of Ottawa and New York, which appear in the previous chapter in Figures 7.8 and 7.20, this individual can be understood to have the same scale x layer as New York, an angle layer approximately equal to that of Ottawa, and a scale y layer which is the same as that for New York except for the south west value which matches that of Ottawa (though whether the result of crossover or mutation would require additional examination).

In generation 2, it is already apparent that the population is exhibiting a denser grid structure.

In generation 3, the individuals all appear dense. The population is beginning to converge towards a specific outcome.
Figure 8.2: Dense Target Demonstration, Generation 1 Individual

Figure 8.3: Breeding Cities User Selection, Dense Demo, Generation 2
Figure 8.4: Breeding Cities User Selection, Dense Demo, Generation 3

Figure 8.5: Breeding Cities User Selection, Dense Demo, Generation 4
In generation 4, one individual appears to be uniformly dense and is selected. Selecting only one individual is permissible, but means that future individuals will be created by performing crossover on an individual and itself, which greatly narrows the gene pool. At this stage, mutation also becomes more important in maintaining diversity in the population.

![Figure 8.6: Breeding Cities User Selection, Dense Demo, Generation 5](image)

Having selected only one individual in generation 4, the population in generation 5 appears much more uniform. One of the individuals is selected as the final product of the evolution, and is shown in detail in Figure 8.7. It can be seen that the evolution did find the most dense outcome possible, hitting the 1.0 ceiling on the scale x and y layers. This particular individual also had approximately the same angle as the Ottawa design.

### 8.2.2 Sparse Target

In this experiment, individuals are selected which appear most sparse. The experiment is run with the same random seed as the dense experiment, meaning that the first generation is the same, though different selections are made which will affect future generations. Images from each generation are shown in Figures 8.8 to 8.12.
Figure 8.7: Dense Target Demonstration, Generation 5 Individual

Figure 8.8: Breeding Cities User Selection, Sparse Demo, Generation 1
In generation 1, several designs are selected, several of which appear very similar to the original New York design.

Figure 8.9: Breeding Cities User Selection, Sparse Demo, Generation 2

In generation 2, it is apparent that the grid densities are becoming more sparse. Given the initial selections in generation 1, all individuals have angles similar to the initial New York design.

In generation 3, most individuals appear to have larger enclosures.

In generation 4, the system is beginning to converge towards a particular outcome. It should be noted that these ‘sparse’ grids all appear to be approximately the size of the larger enclosures within the original New York design.

The population in generation 5 appears similar to that in generation 4. The population will likely take many generation to evolve larger enclosures. This is due to the fact that the only genetic operator that can change leaf values is Gaussian Leaf Mutation. Keeping the probability for this operator low means that more of the original genetic material will remain from one generation to the next (though it may be recombined differently). Larger values will allow the system to move to areas of the solution space that are unreachable with the other genetic operators alone.
Figure 8.10: Breeding Cities User Selection, Sparse Demo, Generation 3

Figure 8.11: Breeding Cities User Selection, Sparse Demo, Generation 4
One of the individuals is selected in generation 5 as the final product of the evolution. It is shown in more detail in Figure 8.13. Though the result is noticeably more sparse than the final result from the dense target demonstration, it still has a scale x value of 0.95 and scale y value of 0.94, which are very high in the range of possible values [0,1]. In the initial Ottawa and New York design genotypes, 0.95 was the lowest leaf value present (see New York’s scale x layer). Therefore, within 5 generations, the system moved to the lowest leaf value of the initial genetic space, and mutated beyond it slightly.

In these experiments, the population quickly converged towards a common target. This was desirable given the small number of generations. It should be noted that there are two methods available to the user to increase diversity. First, the user can choose more individuals at each generation, including individuals with different characteristics. Selecting only one or two in a given generation will quickly diminish the population’s diversity (for better or worse). A second method, previously mentioned, is to use higher parameter values for mutation (particularly for Gaussian Leaf Mutation).
8.3 Detailed Example

In this section, an example is presented which shows the specific effect of recombining the genetic material from the Ottawa and New York designs into a new design. The examples show the effect of the crossover mechanism. The same could be done with mutation operators, but that is left for future work.

This example employs Basic Crossover. Similar results would be found with Weak Context-Preserving Crossover, with the minor added restriction placed on which swap points are permissible. This example was created manually, but accurately reflects the actions of the crossover operator. The crossover operators, discussed in detail in 2.3.2 and 4.4.1, operate on each layer independently. In this example, the effects of crossover on each layer individually are shown in addition to the final composite effect. In actual crossover, these layer-specific crossover individuals do not arise, as the layers are all crossed-over before the crossover operation is complete and the genotype expressed.

The genotypes for Ottawa and New York discovered in the previous chapter are shown with swap points indicated in Figures 8.14 and 8.15.
The effect of this crossover on the scale x layer alone would produce the two individuals shown in Figures 8.16 and 8.17. The effect is more obvious on individual 2, which gains a denser north east region.

The effect of this crossover on the scale y layer alone would produce the two individuals shown in Figures 8.18 and 8.19. This produces visibly wider city blocks in individual 1, and smaller ones in the south west region of individual 2. Perhaps most clearly, individual 1 does intuitively appear to blend the two parent designs.
8.3.3 Angle

The effect of this crossover on the angle layer alone would produce the two individuals shown in Figures 8.20 and 8.21. Individual 2 appears to be based primarily on the segment of New York, with a large area where one might expect Ottawa’s canal’s elbow, with additional density above.

8.3.4 Final

In the actual operation of the system, crossover occurs on all of the layers simultaneously. The final composite effect produces the two individuals shown in Figures 8.22 and 8.23.

These final individuals, which could be produced by the system in its normal operation, do appear to share characteristics of both the Ottawa and New York designs. Through the examination of crossover on each layer, this blending is more fully visualized and understood.
8.4 Summary

In this chapter, the actual process of breeding cities was illustrated. Two demonstrations showed how the system is capable of selectively breeding based on certain traits (grid density) and showed those traits propagated across generations. Additionally, an example was presented which showed the specific formation of city designs through the recombination of the Ottawa and New York designs using crossover.

The demonstrations in this chapter show that the system is capable of breeding new city designs from existing ones. Combined with the previous phase’s ability to discover the genes of imported city designs derived from real city data, it can be seen that CityBreeder enables the rapid, user-guided development of city designs based on the blending of multiple existing designs derived from real cities, thereby answering the problem statement.
Figure 8.18: City Breeding Example, Scale Y Crossover, Individual 1

Figure 8.19: City Breeding Example, Scale Y Crossover, Individual 2
Figure 8.20: City Breeding Example, Angle Crossover, Individual 1

Figure 8.21: City Breeding Example, Angle Crossover, Individual 2
Figure 8.22: City Breeding Example, Complete Crossover, Individual 1

Figure 8.23: City Breeding Example, Complete Crossover, Individual 2
Chapter 9

Conclusions

Cities are complex entities, whose design affects the lives of their many inhabitants. City design is a complex, time-consuming process. Though there have been some procedural advances to speed up this process, none have offered the unique ability of enabling the rapid, user-guided development of city designs based on the blending of multiple existing designs derived from real cities.

In this thesis, CityBreeder was presented, a system based on evolutionary computation which enables this form of city design. In Chapter 4, it was shown how the system’s design and implementation solves the problem statement and enables this ability in principle. Later chapters examined how well it performs, and explored some of its properties scientifically. In Chapter 5, several examples of the genotype-to-phenotype expression mechanism were presented and discussed, exploring its ability to express phenotypes with grids at a variety of densities and angles. In Chapter 6, the first stage of CityBreeder’s pipeline, Discovering City Genes, was tested on a number of simple contrived examples, showing that the system can perform with perfect or very high accuracy. In Chapter 7, this process was tested on city designs derived from real cities. It performed well with designs which exhibit grid-like structure, but had more difficulty with a design that was more irregular. Potential methods of improving the system were also mentioned, and are further discussed in the Future Work section of this chapter. In Chapter 8, the second phase of CityBreeder’s pipeline, Breeding Cities, was explored through demonstrations which created new city designs by breeding together city designs based on Ottawa and New York, and through a more specific example of the recombination of genetic material from these two real city designs.

CityBreeder is not a perfect system; it is one of the first iterations in enabling a unique new method of city design. However, it offers an efficient representation which is well-suited to the city design context, which may also be extended to other problem domains. Additionally, experimental results suggest the system does perform well with a certain set of existing city
designs. The limitations which were encountered are discussed throughout this thesis along with suggestions for how the system may be improved.

With this system, the rapid, user-guided development of city designs based on the blending of multiple existing designs derived from real cities is possible.

9.1 Summary of Contributions

This thesis provides several scientific contributions, which may be summarized as:

- A genetic representation tailored to the city design context, consisting of genotype, phenotype and expression mechanism (Section 4.3)
- A fitness function for the city design context, based on the comparison of categorical maps created from the properties of enclosures within a city design’s road network (Section 4.5)
- A demonstration of the creation of new city designs based on existing designs through the use of evolutionary computation (Chapter 8)
- The discovery of the city genes for several designs based on real cities, and on simple exemplars, with an analysis of the results (Chapters 6 and 7)
- A systematic evaluation of the representation, focussing on expression (Chapter 5), operators (Section 6.3), and fitness function (Section 6.2)

9.2 Future Work

While this thesis demonstrates that CityBreeder answers the problem statement, there is a variety of potential future work, which may be divided into: improvements, additional experimentation, extensions within the city design context, and extensions to other contexts.

9.2.1 Improvements

CityBreeder was shown to discover the genes of several city designs based on real cities, and performed well with Ottawa and New York. However, it had more difficulty with Paris. Future improvements might adjust the expression process in several possible ways to improve
performance. One option would be to add another layer which reflects a measure of non-grid-like properties, and add another step in the expression mechanism to adjust the grid at those locations. The inclusion of a ‘tortuosity’ measure (used in [5]) for roads could also serve to produce more organic road segments. Alternatively, a more fundamental change could be made to create radial (and other) structures during the growth phase of expression. That phase might also be improved through additional ‘snapping’ of nodes to prevent the creation of thin enclosures.

If one did not want to make such large changes, improvement could likely be found by making subtle changes to the rotation phase of expression. Qualitatively, it appears that the rotation phase is less robust than the growth phase. Rotation was found to work well in finding uniform values, but perhaps less so with deeper trees. It was interesting how in some expressed individuals, the rotation enabled phenotypes to be formed by effectively removing particular groups of enclosures by rotating them away, creating uniquely shaped enclosures in their former location (and denser regions where they landed). This effect was observed in Chapter 7 with the evolved Ottawa designs near the elbow in the canal. However, rotation might be improved by preserving edge connections between nodes that are shared between rotation layers before and after rotation. In the present system, groups of enclosures are rotated on separate layers, essentially disconnecting and duplicating previously shared nodes. This was done because connecting them directly after rotation produced unrealistic road edges. Future work might connect each rotated layer in a more subtle and complex manner.

9.2.2 Additional Experimentation

A variety of different experiments were presented in this thesis. Because of that variety, there is room for future work to conduct more experimentation to explore aspects of the system more deeply. Additional experimentation could focus on more thoroughly exploring the parameter space, including operator probability values, the angle contiguity window value, and population size. Experiments could be conducted to evaluate run-time, though this might require an implementation whose focus is software engineering. Experiments could also be conducted that measure genotype similarity over time, as opposed to the included experiments which consider phenotype similarity (as is traditionally done in Evolutionary
Computation). More expression examples could be created to more fully explore how expressive and robust the expression mechanism is.

While the preceding experiments are of scientific interest, the most interesting future experiments might include experiments with larger city segments, perhaps including entire cities, and more experiments with the city breeding process. Additional experiments with the interactive city breeding process could gather subjective feedback from urban planners and other interested parties who might potentially want to use the software. Alternatively, additional quantitative data could be gathered by measuring the fitness of evolved individuals during the breeding process, which is possible if the user is given a specific target to breed ‘towards.’ This later possibility was implemented in an early iteration of CityBreeder, but was later removed during the software development process.

9.2.3 Extensions within the City Design Context

Some of the potential extensions within the city design context have already been mentioned: improvements to better reflect radial and other non-grid structures, and additional experiments which include larger city segments such as entire cities. However, other extensions within the city design context exist. The next logical extension might be to add 3d into the representation. This was partially implemented in a very early iteration of CityBreeder’s development, but was ultimately removed, largely because of the difficulty in obtaining 3d data for real cities (combined with the problem statement’s requirement that the designs be based on real cities). Using artificial 3d data, or perhaps in the future, given better availability of this data, it is natural that city designs could be evolved which blend building height data from real cities as well. Though more challenging, future work could also blend architectural styles. In the related work, multiple approaches to procedural city design create 3d structures around a given a road layout. The most basic approaches include simple extrusion to a given height, while more complex approaches involve grammars and generate architecture. The first iteration of this extension to CityBreeder may be to add one more layer for height, then, in the expression mechanism merely extrude the enclosure (or a padded interior region) to the corresponding height as indicated in the genotype later.

Given the system’s broad, modular representation, other city features could also be modelled through the addition of another layer in the genotype and a new phase in the expression
mechanism. Any of the features discussed earlier in 2.2.2 could be modelled.

9.2.4 Extensions to Other Contexts

In this thesis, a system is presented which is tailored to the city design context. However, the representation is modular and extensible such that it might be applied to other contexts. One possibility is art and image manipulation: ‘breeding’ artistic images based on underlying genetics. The techniques offered in this approach might also be applied to other contexts whose individuals can be represented in a manner similar to the phenotype offered here. Specifically, it is well-suited to contexts whose phenotypes possess spatial properties. Applications with fingerprint identification, or computer vision and edge detection might be applicable.

As more 3d and visual elements are incorporated, this work will be of increasing interest to some of the other communities mentioned in 2.2.4, such as visual effects artists and game level designers.

The possibilities for using CityBreeder to breed new, interesting city designs, and its possible extensions are limited only by one’s imagination.
Bibliography


Appendix A Additional Canadian GIS Data Sources


