Computer Generation and Classification of Music through Operations Research Methods

Dorien Herremans

Promotor: prof. dr. Kenneth Sörensen

Proefschrift voorgedragen tot het behalen van de graad van doctor in de Toegepaste Economische Wetenschappen

Faculteit Toegepaste Economische Wetenschappen - Antwerpen, 2014

Faculty of Applied Economics

Dissertation

COMPOSE ≡ COMPUTE

Computer Generation and Classification of Music through Operations Research Methods

Thesis submitted in order to obtain the degree of Doctor in Applied Economics

Author: Dorien Herremans

Supervisor: Prof. dr. Kenneth Sørensen

December, 2014
Promotor
Prof. dr. Kenneth Sörensen (University of Antwerp)

Members of the Examination Committee
Prof. dr. David Martens (University of Antwerp)
Prof. dr. Trijntje Cornelisens (University of Antwerp)
Prof. dr. Ann De Schepper (University of Antwerp)
Prof. dr. Dirk Moelants (University of Ghent)
Prof. dr. Elaine Chew (Queen Mary University of London)
Prof. dr. Darrell Conklin (University of the Basque Country UPV/EHU and IKERBASQUE)

Cover design: University of Antwerp
Printing and binding: Universitas, Antwerp

© Copyright 2014, Dorien Herremans

All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage and retrieval system, without permission in writing from the author.
“A good thesis topic is one that you still enjoy explaining to strangers in a café at midnight.”¹ According to this definition I have definitely made the right choice. Despite the hard work that has been put into this thesis, I have enjoyed every day that I could work whilst combining two of my passions, being music and technology.

First and foremost, I would like to thank Kenneth Sørensen for supporting me throughout the entire PhD process and giving me the freedom to work on the topic that I had chosen, however exotic it might have seemed in the beginning. His encouragement, advice and knowledge helped me along the way more than anybody else’s.

A thank you also goes to David Martens for guiding me through the fascinating world of data mining. To Johan Springael, for the many interesting discussions. And to Trijntje Cornelissens for allowing me to combine my research with educational activities and the many interesting talks we had.

I have been very fortunate to have a group of fun and supportive colleagues around me. So a big thank you to everybody on the fifth floor. Marco and Pablo, for making the office seem like a home. A special thank you to Marco for sharing some of your bug fixing magic with me. Daniel, for the many discussions whilst enjoying an Indian meal together. Christine, Annelies, Luca, Jochen, Julie, Christof, . . . for making my non-coffee breaks so enjoyable. Also a big thank you to Darrell Conklin for inviting me to come and work in San Sebastián for two months on the extremely interesting Lrn2Cre8 project. All the members of the Lnr2Cre8 project for the great ideas and talks during meetings, at bars and hikes. My fascination for music informatics only grew by this experience.

¹ Told to me by a friend at midnight in a café.
A PhD is not only written in the office. During the longer and more stressful days, I was thankful for the support from my family, my mum, dad, grandma and my good friends Dimi and Jantine for being part of my extended family and being there when I needed them. One special person for making the last months before finishing the PhD extra adventurous. My housemates for providing a warm home and relaxing environment. Wouter, Maarten, Chloë, Evi, Ann, Marlene, Lucy,... Thanks!

And last but certainly not least, a thank you to all of the members of my jury for being part of this process. Your interest and support is much valued.
# OVERVIEW

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>i</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>ix</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>xii</td>
</tr>
<tr>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td><strong>Part 1</strong> MUSIC GENERATION WITH A RULE-BASED OBJECTIVE FUNCTION</td>
<td>7</td>
</tr>
<tr>
<td>1 COMPOSING FIRST SPECIES COUNTERPOINT WITH VNS</td>
<td>9</td>
</tr>
<tr>
<td>2 COMPOSING FIFTH SPECIES COUNTERPOINT WITH VNS</td>
<td>49</td>
</tr>
<tr>
<td>3 FUX, AN ANDROID APP THAT GENERATES COUNTERPOINT</td>
<td>67</td>
</tr>
<tr>
<td><strong>Part 2</strong> MUSIC GENERATION WITH MACHINE LEARNING</td>
<td>79</td>
</tr>
<tr>
<td>4 LOOKING INTO THE MINDS OF BACH, HAYDN AND BEETHOVEN</td>
<td>81</td>
</tr>
<tr>
<td>5 SAMPLING EXTEMA FROM STATISTICAL MODELS</td>
<td>109</td>
</tr>
<tr>
<td>6 GENERATING STRUCTURED MUSIC USING MARKOV MODELS</td>
<td>125</td>
</tr>
<tr>
<td><strong>Part 3</strong> DANCE HIT PREDICTION</td>
<td>151</td>
</tr>
<tr>
<td>7 DANCE HIT SONG PREDICTION</td>
<td>153</td>
</tr>
<tr>
<td><strong>Conclusions</strong></td>
<td>183</td>
</tr>
</tbody>
</table>
OVERVIEW

DUTCH SUMMARY 187

A DETAILED BREAKDOWN OF OBJECTIVE FUNCTION FOR FIRST SPECIES COUNTERPOINT 191

B DETAILED BREAKDOWN OF THE OBJECTIVE FUNCTION FOR FIFTH SPECIES COUNTERPOINT 199

C LIST OF PUBLICATIONS 207

BIBLIOGRAPHY 209

INDEX 230
# CONTENTS

<table>
<thead>
<tr>
<th>ACKNOWLEDGEMENTS</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>ix</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>xii</td>
</tr>
<tr>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
</tbody>
</table>

## Part 1  Music generation with a rule-based objective function

1  Composing first species counterpoint with VNS
   1.1 Introduction ........................................... 9
   1.2 Quantifying musical quality .......................... 10
   1.3 Variable neighbourhood search to generate counterpoint fragments .................................................. 17
   1.3.1 Components of the VNS algorithm ................. 20
   1.3.2 General outline of the implemented VNS algorithm ...... 22
   1.4 Implementation ......................................... 26
   1.5 Experiments ............................................. 28
   1.5.1 VNS parameter optimisation ....................... 29
   1.5.2 VNS versus random search ......................... 40
   1.5.3 VNS versus GA ...................................... 42
   1.6 Conclusions ............................................ 46

2  Composing fifth species counterpoint with VNS
   2.1 Introduction ........................................... 49
   2.2 From first to fifth species ............................ 50
   2.3 Quantifying musical quality ........................... 51
   2.4 Variable neighbourhood search ........................ 53
   2.5 Architecture and implementation .................... 55
   2.6 Experiments ............................................. 57
   2.7 Conclusions ............................................ 59

3  Fux, an Android app that generates counterpoint
   3.1 Introduction ........................................... 67
   3.2 From Optimuse to FuX .................................. 68
6.2.1 Cycles and patterns ........................................ 130
6.2.2 Realizing and evaluating cyclic patterns .......... 131
6.2.3 Compound cyclic patterns ............................ 132
6.3 Using Markov models within evaluation metrics .... 133
  6.3.1 High probability sequences (XE) .................. 134
  6.3.2 Minimal distance between TM of model and solution (DI) 135
  6.3.3 Delta cross-entropy (DE) ........................ 135
  6.3.4 Information contour (i) .......................... 137
  6.3.5 Unwords (u) ........................................ 138
6.4 Variable neighbourhood search .......................... 139
6.5 Results ................................................. 141
  6.5.1 Training data and Markov model ................ 141
  6.5.2 Musical results .................................. 142
6.6 Conclusions ........................................... 148

Part 3  DANCE HIT PREDICTION 151

7 DANCE HIT SONG PREDICTION 153
  7.1 Introduction .......................................... 154
  7.2 Dataset .............................................. 156
    7.2.1 Hit listings .................................. 156
    7.2.2 Feature extraction and calculation ............ 158
  7.3 Evolution over time .................................. 161
  7.4 Dance hit prediction ................................ 163
    7.4.1 Experiment setup ............................... 163
    7.4.2 Preprocessing ................................ 166
  7.5 Classification techniques ............................ 168
    7.5.1 C4.5 tree ....................................... 170
    7.5.2 RIPPER ruleset ................................ 170
    7.5.3 Naive Bayes ................................... 172
    7.5.4 Logistic regression ............................. 173
    7.5.5 Support vector machines ....................... 174
  7.6 Results ............................................... 177
    7.6.1 Full experiment with cross-validation ....... 177
    7.6.2 Experiment with out-of-time test set .......... 181
  7.7 Conclusions ......................................... 181

CONCLUSIONS 183
CONTENTS

DUTCH SUMMARY 187
A DETAILED BREAKDOWN OF OBJECTIVE FUNCTION FOR FIRST SPECIES COUNTERPOINT 191
B DETAILED BREAKDOWN OF THE OBJECTIVE FUNCTION FOR FIFTH SPECIES COUNTERPOINT 199
C LIST OF PUBLICATIONS 207

BIBLIOGRAPHY 209
INDEX 230
**LIST OF FIGURES**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Cantus firmus and counterpoint example with objective score ( f(s) = 1.589 )</td>
<td>20</td>
</tr>
<tr>
<td>1.2</td>
<td>A perturbation move is used to “jump” out of a local optimum</td>
<td>21</td>
</tr>
<tr>
<td>1.3</td>
<td>Moves</td>
<td>24</td>
</tr>
<tr>
<td>1.4</td>
<td>Flowchart of the developed VNS algorithm</td>
<td>27</td>
</tr>
<tr>
<td>1.5</td>
<td>MuseScore</td>
<td>28</td>
</tr>
<tr>
<td>1.6</td>
<td>Means plots CF</td>
<td>34</td>
</tr>
<tr>
<td>1.7</td>
<td>Mean plots CP</td>
<td>35</td>
</tr>
<tr>
<td>1.8</td>
<td>Evolution of the VNS with optimal parameter settings</td>
<td>39</td>
</tr>
<tr>
<td>1.9</td>
<td>Comparison of VNS and random search</td>
<td>41</td>
</tr>
<tr>
<td>1.10</td>
<td>Comparison of VNS and GA</td>
<td>45</td>
</tr>
<tr>
<td>1.11</td>
<td>Generated counterpoint with score of 0.371394</td>
<td>46</td>
</tr>
<tr>
<td>2.1</td>
<td>First species counterpoint fragment (generated by Optimuse)</td>
<td>52</td>
</tr>
<tr>
<td>2.2</td>
<td>Fifth species counterpoint fragment (Salzer and Schacter, 1969)</td>
<td>52</td>
</tr>
<tr>
<td>2.3</td>
<td>Optimuse plugin in MuseScore</td>
<td>58</td>
</tr>
<tr>
<td>2.4</td>
<td>Mean plots CP</td>
<td>63</td>
</tr>
<tr>
<td>2.5</td>
<td>Evolution over time with optimal parameter settings</td>
<td>64</td>
</tr>
<tr>
<td>2.6</td>
<td>Fifth species counterpoint fragment (Optimuse)</td>
<td>65</td>
</tr>
<tr>
<td>3.1</td>
<td>Overview of the developed VNS Algorithm</td>
<td>71</td>
</tr>
<tr>
<td>3.2</td>
<td>Evolution of the objective function over time</td>
<td>75</td>
</tr>
<tr>
<td>3.3</td>
<td>FuX 1.0 user interface</td>
<td>77</td>
</tr>
<tr>
<td>4.1</td>
<td>Ruleset</td>
<td>92</td>
</tr>
<tr>
<td>4.2</td>
<td>Timeline of the composers Bach, Beethoven and Haydn (Greene, 1985)</td>
<td>93</td>
</tr>
<tr>
<td>4.3</td>
<td>C4.5 decision tree</td>
<td>94</td>
</tr>
<tr>
<td>4.4</td>
<td>Probability that a piece is composed by composer ( i )</td>
<td>98</td>
</tr>
<tr>
<td>Figure</td>
<td>Title</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>4.5</td>
<td>Illustration of SVM optimization of the margin in the feature space.</td>
<td>99</td>
</tr>
<tr>
<td>4.6</td>
<td>ROC curves of the best performing models</td>
<td>101</td>
</tr>
<tr>
<td>4.7</td>
<td>User interface of FuX</td>
<td>103</td>
</tr>
<tr>
<td>4.8</td>
<td>Evolution of solution quality over time</td>
<td>105</td>
</tr>
<tr>
<td>4.9</td>
<td>A generated fragment with high probability for Bach (99.94%)</td>
<td>106</td>
</tr>
<tr>
<td>4.10</td>
<td>A generated fragment with high probability for Beethoven (99.64%)</td>
<td>107</td>
</tr>
<tr>
<td>4.11</td>
<td>A generated fragment with high probability for Haydn (99.31%)</td>
<td>107</td>
</tr>
<tr>
<td>5.1</td>
<td>First species counterpoint example (Salzer and Schachter, 1969) and its dyad representation</td>
<td>112</td>
</tr>
<tr>
<td>5.2</td>
<td>Features (on the arrows) derived from two consecutive dyads a and b (bottom)</td>
<td>113</td>
</tr>
<tr>
<td>5.3</td>
<td>The probabilistic dependencies in the vertical viewpoint model</td>
<td>113</td>
</tr>
<tr>
<td>5.4</td>
<td>Overview of the VNS</td>
<td>117</td>
</tr>
<tr>
<td>5.5</td>
<td>The distribution of cross-entropy according to random walk, over $10^7$ iterations</td>
<td>120</td>
</tr>
<tr>
<td>5.6</td>
<td>Evolution of the best fragment found</td>
<td>121</td>
</tr>
<tr>
<td>5.7</td>
<td>Evolution of the best fragment found zoomed in on the beginning of the runs</td>
<td>122</td>
</tr>
<tr>
<td>6.1</td>
<td>Assignment of fingers to strings on the bagana and their closest Western pitches (in letter notation)</td>
<td>130</td>
</tr>
<tr>
<td>6.2</td>
<td>Tew Semagn Hagere by Alemu Aga, as transcribed by Weisser (2005)</td>
<td>131</td>
</tr>
<tr>
<td>6.3</td>
<td>Histogram of cross-entropy values of the corpus</td>
<td>136</td>
</tr>
<tr>
<td>6.4</td>
<td>Overview of the VNS.</td>
<td>140</td>
</tr>
<tr>
<td>6.5</td>
<td>Evolution of cross-entropy and distance of transition matrices over time</td>
<td>144</td>
</tr>
<tr>
<td>6.6</td>
<td>Musical output by using the three main evaluation metrics (XE, DI and DE)</td>
<td>145</td>
</tr>
<tr>
<td>6.7</td>
<td>Musical output by using the main three evaluation metrics combined with unwords (u)</td>
<td>146</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>6.8</td>
<td>Musical output by using the main three evaluation metrics combined with information contour (i)</td>
<td>147</td>
</tr>
<tr>
<td>7.1</td>
<td>Motion chart visualising evolution of dance hits from 1985 until 2013</td>
<td>162</td>
</tr>
<tr>
<td>7.2</td>
<td>Evolution over time of selected characteristics of top 10 songs</td>
<td>164</td>
</tr>
<tr>
<td>7.3</td>
<td>Frequency distribution of the beats per minute feature over the OCC dataset</td>
<td>165</td>
</tr>
<tr>
<td>7.4</td>
<td>Flow chart of the experimental setup</td>
<td>167</td>
</tr>
<tr>
<td>7.5</td>
<td>Class distribution</td>
<td>168</td>
</tr>
<tr>
<td>7.6</td>
<td>C4.5 decision tree</td>
<td>171</td>
</tr>
<tr>
<td>7.7</td>
<td>Probability that song $i$ is a dance hit</td>
<td>173</td>
</tr>
<tr>
<td>7.8</td>
<td>Illustration of SVM optimization of the margin in the feature space</td>
<td>175</td>
</tr>
<tr>
<td>7.9</td>
<td>ROC for logistic regression</td>
<td>180</td>
</tr>
<tr>
<td>7.10</td>
<td>Class distribution of the split training and test sets</td>
<td>180</td>
</tr>
</tbody>
</table>
# List of Tables

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Summary of the motivations and domains of CAC, as adapted from Pearce et al. (2002)</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1.1</td>
<td>Neighbourhoods</td>
<td>24</td>
</tr>
<tr>
<td>Table 1.2</td>
<td>Parameters of the VNS</td>
<td>30</td>
</tr>
<tr>
<td>Table 1.3</td>
<td>Model without interactions (CF) - Summary of Fit</td>
<td>30</td>
</tr>
<tr>
<td>Table 1.4</td>
<td>Multi-Way ANOVA model without interactions (CF)</td>
<td>31</td>
</tr>
<tr>
<td>Table 1.5</td>
<td>Model with interactions (CF) - Summary of Fit</td>
<td>31</td>
</tr>
<tr>
<td>Table 1.6</td>
<td>Multi-Way ANOVA model with interactions (CF) (extract)</td>
<td>33</td>
</tr>
<tr>
<td>Table 1.7</td>
<td>Multi-Way ANOVA model for Time</td>
<td>36</td>
</tr>
<tr>
<td>Table 1.8</td>
<td>Model with interaction effects (CP) - Summary of Fit</td>
<td>37</td>
</tr>
<tr>
<td>Table 1.9</td>
<td>Multi-Way ANOVA model with interaction effects (CP)</td>
<td>37</td>
</tr>
<tr>
<td>Table 1.10</td>
<td>Best parameter settings</td>
<td>38</td>
</tr>
<tr>
<td>Table 1.11</td>
<td>Parameters of the GA</td>
<td>42</td>
</tr>
<tr>
<td>Table 1.12</td>
<td>Multi-Way ANOVA model with interactions (CP)</td>
<td>43</td>
</tr>
<tr>
<td>Table 1.13</td>
<td>Optimal Parameter settings for the GA</td>
<td>43</td>
</tr>
<tr>
<td>Table 1.14</td>
<td>A comparison of the mean running time for the VNS and GA</td>
<td>44</td>
</tr>
<tr>
<td>Table 2.1</td>
<td>Feasibility criteria</td>
<td>55</td>
</tr>
<tr>
<td>Table 2.2</td>
<td>Properties of the Note object</td>
<td>57</td>
</tr>
<tr>
<td>Table 2.3</td>
<td>Parameters</td>
<td>59</td>
</tr>
<tr>
<td>Table 2.4</td>
<td>Multi-Way ANOVA model with interactions - Summary of Fit</td>
<td>60</td>
</tr>
<tr>
<td>Table 2.5</td>
<td>Multi-Way ANOVA model with interactions</td>
<td>61</td>
</tr>
<tr>
<td>Table 2.6</td>
<td>Best parameters</td>
<td>62</td>
</tr>
<tr>
<td>Table 3.1</td>
<td>neighbourhoods</td>
<td>70</td>
</tr>
<tr>
<td>Table 3.2</td>
<td>Multithreading</td>
<td>73</td>
</tr>
<tr>
<td>Table 4.1</td>
<td>Dataset</td>
<td>86</td>
</tr>
<tr>
<td>Table 4.2</td>
<td>Analysed features</td>
<td>87</td>
</tr>
<tr>
<td>Table 4.3</td>
<td>Evaluation of the models with 10-fold cross-validation</td>
<td>90</td>
</tr>
<tr>
<td>Table 4.4</td>
<td>Confusion matrix for RIPPER</td>
<td>92</td>
</tr>
</tbody>
</table>
Table 4.5  Confusion matrix for C4.5 ............................. 95
Table 4.6  Confusion matrix logistic regression ................. 98
Table 4.7  Confusion matrix for support vector machines ....... 100
Table 5.1  Average and best results of 100 runs after 30 million
transition matrix lookups ...................................... 120
Table 6.1  The set of unwords that were found in the bagana corpus139
Table 6.2  Transition matrix based on the bagana corpus; finger
numbers as indices, and corresponding pitch class
names (Tezeta scale) in brackets ................................. 142
Table 6.3  General characteristics of the generated music
displayed in Figures 6.6, 6.7 and 6.8 ......................... 143
Table 7.1  Hit listings overview .................................... 157
Table 7.2  Example of hit listings before adding musical features 157
Table 7.3  Datasets used for the dance hit prediction model . . . 166
Table 7.4  The most commonly occurring features in D1, D2 and
D3 after FS ..................................................... 169
Table 7.5  RIPPER ruleset ........................................... 171
Table 7.6  Results with 10-fold validation (accuracy) ............ 178
Table 7.7  Results for 10 runs with 10-fold validation (AUC) ... 178
Table 7.8  Results for 10 runs on D1 (FS) with 10-fold cross-
validation compared with the split test set .................. 179
Table 7.9  Confusion matrix logistic regression ................... 181
Table 7.10 Probability of recent dance songs being a top 10 hit
according to the logistic regression model (D1) ............ 182
INTRODUCTION

With the birth of the digital computer came the question of using it for musical composition. Ever since the 50s computers have been used to compose music (Pinkerton, 1956; Brooks et al., 1957). With the rise of the personal computer at the end of the last millennium, automated music composition systems have been developed by a wide range of people, among them musicians, psychologists, computer scientists and philosophers. These people have different objectives in mind, which Pearce et al. (2002) categorises into four classes (see Table 1). The first motivation for computer automated composition (CAC) systems is that the composer sees these systems as an idiosyncratic extension to his or her own compositional processes. As Cope (1991) stated: “EMI was conceived... as the direct result of a composer’s block. Feeling I needed a composing partner, I turned to computers believing that computational procedures would cure my block”. Secondly, they can be seen as tools to aid a composer during the compositional processes. Researchers at IRCAM often work together with composers, aiming to build visual composition systems such as the one described by Assayag et al. (1999). Thirdly, automatic composition systems can implement theories of musical styles. The expert system CHORAL developed by Ebcioğlu (1990) can harmonize four-part chorales in the style of J.S. Bach based on 350 style rules. A fourth motivation is that they can implement cognitive theories of the processes supporting compositional expertise. This was clearly Johnson-Laird (1991)’s motivation as he intended to develop a theory of “what the mind has to compute in order to produce an acceptable improvisation”.

This dissertation compiles the main results of the research that I have conducted during the last four years. It focusses mainly on the use of techniques from operations research applied to the domain of music composition and classification. Operations research (OR) is a versatile field that focusses on
Table 1: Summary of the motivations and domains of CAC, as adapted from Pearce et al. (2002)

<table>
<thead>
<tr>
<th>Domain</th>
<th>Activity</th>
<th>Motivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composition</td>
<td>Algorithmic composition</td>
<td>Expansion of compositional repertoire</td>
</tr>
<tr>
<td>Software engineering</td>
<td>Design of compositional tools</td>
<td>Development of tools for composers</td>
</tr>
<tr>
<td>Musicology</td>
<td>Computational modelling of musical styles</td>
<td>Proposal and evaluation of theories of musical styles</td>
</tr>
<tr>
<td>Cognitive science</td>
<td>Computational modelling of music cognition</td>
<td>Proposal and evaluation of cognitive theories of musical composition</td>
</tr>
</tbody>
</table>

mathematical modelling of complex problems to find optimal or feasible solutions. These techniques are applied in a wide range of different domains and have a lot to offer in terms of solving problems in music, composition, analysis, and performance (Chew, 2008).

The compositional music systems described in this dissertation take an interdisciplinary point of view, as they were born from a combination of motivations, i.e., the first three from Table 1.

The first one, artistic motivation, is what inspired the very first thought of creating this thesis and comes from my personal interest in composing. Based on this, music generation systems were created that fall into two major categories. In the first category the algorithm evaluates the generated music based on criteria from music theory and is discussed in Part 1. In Part 2 the focus is shifted towards automatically learning what makes music sound good and using models from machine learning to build evaluation metrics for different styles of music. In the last part, the use of machine learning techniques in the domain of dance hit prediction is explored.

In the first two parts, a powerful variable neighbourhood search algorithm for music generation is developed. This offers a useful tool for other computa-
tional composers, and thus fulfils the second motivation specified by Pearce et al. (2002) by providing a practical compositional tool.

The third motivation, evaluation of musical styles, is clearly present in Part 2 and 3, where the focus lies on using machine learning tools to learn models from music that can be used for either musical style assessment or classification. While the chapters in Part 2 combine machine learning with music generation. The classification task on its own is examined in more detail in the last chapter on dance hit prediction.

**Part 1: Music generation with a rule-based objective function**

This part explores the use of combinatorial optimization techniques, more specifically metaheuristics, for the generation of music. The objective function of the algorithm is based on rules that are quantified from music theory.

In Chapter 1 a simplified compositional problem, i.e., first species counterpoint, is used to test and compare the efficiency of a developed variable neighbourhood search algorithm (VNS) and a genetic algorithm (GA). Both algorithms are thoroughly evaluated and their parameters are set after performing a full factorial experiment. An extensive set of rules are quantified and implemented as the objective function.

Each chapter of this thesis is based on a paper, the full list of papers is available in Appendix C. The research in this chapter has given rise to the following scientific publication:


In Chapter 2 the VNS algorithm for generating first species counterpoint is expanded to work with fifth species, a more complex form of music that also includes a rhythmic aspect. An additional large set of rules was quantified from music theory and implemented as evaluation criteria for the quality of the generated music. The following paper was published based on this chapter:
Chapter 3 describes the modification and implementation of this system as an Android app called FuX. This app is able to generate a stream of continuous music that can be played on any Android device. The chapter is based on the following conference paper:


**Part 2: Music generation with machine learning**

In this part, we break free from using a predefined objective function and move towards a machine learned model to evaluate the quality of generated pieces.

In Chapter 4 different classification models are built that can accurately classify pieces between three composers (Bach, Beethoven and Haydn). The best model is integrated in the objective function of the VNS algorithm to allow the generation of music with characteristics of a certain composer. This system is integrated in the FuX Android app. A paper on this topic based on the following working paper has been submitted to a journal:


In Chapter 5 a Markov model is built with the vertical viewpoints method based on first species counterpoint music. The use of the variable neighbourhood search algorithm is evaluated for generating high-probability sequences based on this statistical model. This chapter is based on the following conference paper:
In chapter 6 different ways are examined in which low order Markov models can be used to build quality assessment metrics for optimization algorithms. These metrics are compared and evaluated in an experiment in which structured bagana music (i.e., an Ethiopian lyre) is generated with VNS. Due to the size of many datasets it is often only possible to get rich and reliable statistics for low order models, yet these do not handle structure very well and their output is often very repetitive. A method was proposed that allows the enforcement of structure and repetition within music, thus handling long term coherence with a first order model. The following working paper forms the basis of this chapter:


Part 3: Dance hit prediction

In the final chapter, the power of machine learning on audio data is put to the test and a system for dance hit prediction is built. This system is based on historical chart data combined with basic musical features as well as more advanced features that capture a temporal aspect. A number of different classifiers are used to build and test dance hit prediction models. The resulting best model has a good performance when predicting whether a song is a “top 10” dance hit versus a lower listed position. This chapter is based on the following scientific publication:


As apparent from the above, each chapter is based on an independent publication. The amount of overlap has been reduced as much as possible. However,
INTRODUCTION

a slight amount of overlap has been kept in order to preserve the structure and logical connections within the chapters.
Part 1

MUSIC GENERATION WITH A RULE-BASED OBJECTIVE FUNCTION
COMPOSING FIRST SPECIES COUNTERPOINT MUSICAL SCORES WITH A VARIABLE NEIGHBOURHOOD SEARCH ALGORITHM

This chapter is based on the paper 'D. Herremans, K. Sørensen. 2012. Composing first species counterpoint musical scores with a variable neighbourhood search algorithm. Journal of Mathematics and the Arts. 6(04):169-189.'
In this chapter a variable neighbourhood search (VNS) algorithm is developed that can generate musical fragments consisting of a cantus firmus and a first species counterpoint melody. The objective function of the algorithm is based on a quantification of existing counterpoint rules. The VNS algorithm developed in this chapter is a local search algorithm that starts from a randomly generated melody and improves it by changing one or two notes at a time.

A thorough parametric analysis of the VNS reveals the significance of the algorithm’s parameters on the quality of the composed fragment, as well as their optimal settings. A comparison of the VNS algorithm with a developed genetic algorithm shows that the VNS is more efficient.

The VNS algorithm has been implemented in a user-friendly software environment for composition, called Optimuse. Optimuse allows a user to specify a number of characteristics such as length, key, and mode. Based on this information, Optimuse “composes” both a cantus firmus and a first species counterpoint melody. Alternatively, the user may specify a cantus firmus, and let Optimuse compose an accompanying first species counterpoint melody.

1.1 INTRODUCTION

Computers and music go hand in hand these days: music is stored digitally and often played by electronic instruments. Computer aided composing (CAC) is a relatively new research area that takes the current relationship between music and computers one step further, allowing the computer to aid a composer or even generate an original score. The idea of automatic music generation is not new, and one of the earliest “automatic composition” methods based on randomness is due to Mozart. In his Musikalisches Würfelspiel (Musical Dice Game), a number of small musical fragments are combined by chance to generate a Minuet (Boenn et al., 2009). Other, more modern composers also make use of chance in the composition of their pieces. John Cage’s piece “Reunion” (1968) is performed by playing chess on a chessboard equipped by a photo-recepter. Each move on the chessboard triggers electronic sounds and
thus a different piece is performed each time a game of chess is played (Fetterman, 1996). Another example of chance-inspired music by Cage is “Atlas Eclipticalis”, a piece from 1961, composed by superimposing musical staves on astronomical charts (Burns, 2004). Natural phenomena also inspired composer Charles Dodge. His piece “The Earth’s Magnetic Field” in 1970, is a musical translation of the fluctuations of the earth’s magnetic field (Alpern, 1995). Other examples of compositions based on stochastics are Xenakis’s “Pithoprakta” and “Metastaseis”. These pieces are inspired by natural phenomena such as the flights of starlings and a swarm of bees (Fayers, 2011).

Lejaren Hiller and Leonard Isaacson used the Illiac computer in 1957 to generate the score of a string quartet. The resulting “Illiac Suite” is one of the first musical pieces successfully composed by a computer (Sandred et al., 2009). Hiller and Isaacson simulate the compositional process with a three-step rule-based approach (Alpern, 1995). In the first step, random pitches and rhythms are generated. Then, screening rules determine whether the components of the raw composition are accepted or rejected. Various techniques such as permutation and geometric transformations are applied to improve this compositional base. A set of selection rules finally determines the material to be included in the final composition (Nierhaus, 2009).

Another pioneer in algorithmic composition is Iannis Xenakis. Xenakis uses stochastic methods to aid his compositional process. For example, in “Analogique A’, Markov models are used to control the order of musical sections (Xenakis, 1992). A more extensive overview of techniques of mid-to-late 20th century’s avant-garde composers is given by David Cope in (Cope, 2000).

Some studies have modelled a style with a statistical model and used this to harmonize music. Whorley et al. (2013) use the multiple viewpoints system for four-part harmonization. The harmonization problem was also tackled by Allan (2002), who describes the Bach chorale harmonization problem with hidden Markov models. Paiement et al. (2006) introduce a graphical model for the harmonization of melodies that considers every structural component in the chord notation. Suzuki and Kitahara (2014) explore Bayesian network models that generate four-part harmonies according to the melody of
a soprano voice. Generating music from statistical models will be discussed more in detail in Part 2.

In this chapter an algorithm is developed for automatic composition of first species counterpoint music. It is argued that composing music can—at least partially—be regarded as a combinatorial optimisation problem, in which one or more melodies are searched to adhere to the “rules” of their specific musical style. To this end, these rules need to be formalised and quantified, so that the algorithm can judge whether one automatically composed melody is better than another. The better a melody fits within a certain style, the higher its solution quality will be. Although every musical genre has its own rules, these have generally not been formally written down (Moore, 2001). The main reason for the choice of the counterpoint style, is that in this style there do exist formal, written rules (Fux and Mann, 1971) that can be quantified. This chapter therefore focuses on automatic composition of one or two melodies that adhere to a particular set of counterpoint rules. The algorithm could then in principle be adapted to compose melodies from other styles, using rules that have been data-mined from a musical database. This will be explored in Part 2.

Composing music is computationally complex, especially since the number of possible melodies increases exponentially with the length (number of notes) of the melody to compose. For instance, a musical fragment consisting of 16 notes, without rhythm, in which each note can take on 14 different pitches, already has $14^{16}$ (or roughly 2.18 quintillion) possible note combinations. This makes exact methods like exhaustive enumeration practically impossible to use. Heuristic or metaheuristic optimisation algorithms therefore present the most promising approaches to generate high-quality melodies in a reasonable amount of time. Contrary to exact methods, (meta)heuristics do not necessarily return the optimal (i.e., best possible) solution (Blum and Roli, 2003), but use a variety of strategies, some seemingly unrelated to optimisation (such as natural selection or the cooling of a crystalline solid), to find good solutions. A large number of metaheuristics has been proposed, which can be roughly divided into three categories. Local search metaheuristics (tabu search, variable neighbourhood search, iterated local search, . . . ) iteratively improve a single solution. Constructive metaheuristics (GRASP, ant colony optimisation, . . . )
1.1. INTRODUCTION

build a solution from its constituent parts. Population-based metaheuristics (genetic/evolutionary algorithms, path relinking, . . . ) maintain a set (usually called the population) of solutions and combine solutions from this set into new ones (Sörensen and Glover, 2013).

Most methods for CAC found in the literature belong to the class of population-based, more specifically evolutionary, algorithms (Nierhaus, 2009) that find inspiration in the process of natural evolution. Their popularity is largely due to the fact that they do not rely on a specific problem structure or problem-specific knowledge (Horner and Goldberg, 1991). The use of other types (constructive and/or local search) of metaheuristics has remained relatively unexplored.

The first published record of a genetic algorithm used for computer aided composition is by Horner and Goldberg (1991). Their genetic algorithm (GA) is applied for thematic bridging, i.e., transforming an initial musical fragment to a final fragment over a specified duration. An extensive overview of the applications of genetic algorithms in CAC over the following decade is given by Burton and Vladimirova (1999) and Todd and Werner (1999). The second category of metaheuristics, the constructive metaheuristics, has not received as much attention as the population-based metaheuristics, especially in the domain of CAC. The first ant colony metaheuristic for harmonizing a melody was developed by Geis and Middendorf (2007). Local search techniques, the third category, have been used at IRCAM (Institut de Recherche et Coordination Acoustique/Musique) for solving musical constraint satisfaction problems (Truchet and Codognet, 2004). The potential of local search metaheuristics in the area of CAC remains an interesting area for exploration.

Several CAC methods rely on user input. GenJam (Biles, 2001) uses an evolutionary algorithm to optimise monophonic jazz solos, relative to a given chord progression. The quality of a solo is calculated based on feedback from a human mentor. This creates a bottleneck because the mentor needs to listen to each solo and take the time to evaluate it (Biles, 2003). A comparable human fitness bottleneck also arises in the CONGA system, a rhythmic pattern generator based on an evolutionary algorithm. According to the authors of the system, the lack of a quantifiable fitness function slows down the composing
process and places a psychological burden on the user who listens and has to determine the fitness value (Tokui and Iba, 2000). Horowitz developed interactive genetic algorithm to generate rhythmic patterns. Selection is done based on an objective fitness function, whose target levels are set by the user. However, each new generation of rhythms has to be judged subjectively by the user (Horowitz, 1994).

Other CAC algorithms have attempted to eliminate the human bottleneck and define an objective function that can be automatically calculated. Moroni (Moroni et al., 2000) uses a genetic algorithm to evolve a population of chords. Its fitness criteria are based on physical factors related to melody, harmony and vocal range. Another approach is used by Towsey et al. (2001) to develop a GA that composes Western Diatonic Music based on best practices. Their fitness function is based on 21 features and was constructed from statistics gathered by the analysis of a library of melodies.

Other studies base themselves on rules of existing music theory. Geis and Middendorf (2007) implement a function that indicates how well a given fragment adheres to five harmonic rules of baroque music. This function is then used as the objective function of an ant colony metaheuristic. Orchidée is a multiobjective genetic algorithm that combines musical fragments (from a database) in order to efficiently orchestrate them. The goal of this approach is to find a combination of samples that, when played together, sounds as close as possible to the target timbre (Carpentier et al., 2010).

Notwithstanding the above examples, very little research can be found on the automatic evaluation of a melody. As mentioned, formal, written-down rules for a given musical style typically do not exist. An exception is counterpoint music: the formalised nature of this style is exemplified by the fact that simple rules exist for both a single melody and the harmony between several melodies (Rothgeb, 1975). Counterpoint music starts from a single melody called the cantus firmus (“fixed song”), composed according to a set of melodic rules and adds a second melody (the counterpoint). The counterpoint is composed according to a similar set of melodic rules, but also needs to adhere to a set of harmonic rules, that describe the relationship between the cantus firmus and the counterpoint (Norden, 1969).
One of the earliest attempts to generate counterpoint is due to Lewin (1983). He generates first-species counterpoint using his own “Global Rule” in addition to the standard rules. The counterpoint is generated backwards from the cadence note in order to implement the Golden Rule more easily. Gjerdingen (1988) developed a system called PRAENESTE. This system approaches the problem more from a musician’s point of view instead of that of a programmer. The generation process moves forward in time without the benefit of being able to go back and start over. At each given time, PRAENESTE, selects a melodic pattern from a small number of concrete musical schemata.

Aguilera et al. (2010) use probabilistic logic to generate counterpoint music in C major, based on a given cantus firmus. GPmuse (Polito et al., 1997) implements a genetic algorithm that optimises florid counterpoint in C major, by using a fitness function based on the homework problems described by Fux.

David Cope’s Experiments in Musical Intelligence (EMI) focus on understanding a composer-specific style, by extracting signatures of musical pieces using pattern matching and then generating music with a grammar based system (Papadopoulos and Wiggins, 1999). More recently, Cope developed Gradus, named after Johann Fux’s Gradus at Parnassum (Fux and Mann, 1971), which can compose first species counterpoint, given a cantus firmus. Gradus analyses a set of first species counterpoint examples and learns the best setting for 6 general counterpoint goals or rules. These goals are used to sequentially generate the piece (Cope, 2004).

Strasheela is a generic constraint programming system for composing music. Anders (2007) uses the Strasheela system to compose first species counterpoint based on 6 rules. Other constraint programming languages, such as PWConstraints developed at IRCAM can be used to generate counterpoint, provided that the user inputs the correct rules (Assayag et al., 1999). A more complete overview of constraint programming systems applied to music generation systems is given by Henz et al. (1996).

Counterpoint also includes rules for four-part music. Based on these rules, Ebcioğlu (1988) has developed a knowledge-based expert system for
the harmonisation of four-part Bach chorales. This system uses a generate-and-test method with intelligent backtracking. De Prisco et al. (2010) trained three neural networks in parallel, using Bach chorales, in order to harmonize a bass line. For each note in the bass line, their algorithms try to find a three or four note chord. The output of the neural network was compared to the original harmonization made by Bach and tends to coincide in 85-90% of the cases. McIntyre (1994) harmonizes similar four-part Bach chorales, by making use of a genetic algorithm. Donnelly and Sheppard (2011) also develop a genetic algorithm that evolves four-part harmonic music. A similar problem was tackled by Phon-Amnuaisuk et al. (1999) using a fitness function based on a subset of the four-part counterpoint rules. Phon-Amnuaisuk and Wiggins (1999) also compare a rule-based system with the genetic algorithm for harmonizing four-part monophonic tonal music. They discovered that the rule-based system delivers much better output than the genetic algorithm. Their conclusion is that “The output of any system is fundamentally dependent on the overall knowledge that the system (explicitly and implicitly) possesses” (Phon-Amnuaisuk and Wiggins, 1999). This supports the previous claim that metaheuristics that make use of problem-specific knowledge might be more efficient than a black-box genetic algorithm. A more extensive overview of algorithmic composing is given by Nierhaus (2009). To the authors’ knowledge, a local search metaheuristic such as variable neighbourhood search has not yet been used to generate counterpoint music.

Although other studies, including the aforementioned, have used Fux’s rules in order to calculate the counterpoint-quality of a musical fragment, the exact way in which they are quantified is usually not mentioned in detail. Some studies only focus on a few of the most important rules instead of looking at the entire theory (Anders, 2007). The next section will discuss the Fuxian rules for first species counterpoint and how they have been quantified to determine the “quality” (i.e., adherence to the rules) of a fragment consisting of a cantus firmus and a counterpoint melody. This quality score will be used as the objective function of a local search metaheuristic, that is developed in Section 1.3. The practical implementation of the developed algorithm is described in Section 1.4. In Section 1.5, a thorough parametric analysis is performed and the efficiency of the algorithm is compared to a random search
and a genetic algorithm. Final conclusions and future research opportunities are discussed in Section 1.6.

1.2 QUANTIFYING MUSICAL QUALITY

This chapter focuses on a specific type of polyphonic classical music called first species counterpoint. A polyphonic musical fragment consists of two or more voices, also called parts or melodies. The term “counterpoint” refers to the relationship between those melodies. The complexities that arise from playing different notes at the same time has given rise to a very restrictive and formalised set of rules on how to compose polyphonic music.

The species counterpoint rules written by Johann Fux in 1725 are one of the most restrictive sets of rules for composing renaissance music. This system was originally developed as a pedagogical tool, for students learning how to compose. It consists of five species or levels (first, second, third, fourth and the most complex is called florid counterpoint) that are taught in sequence. Each species adds more complexity to the music, e.g., more notes per part, or different rhythmical structure. The Fuxian counterpoint rules are foundational in music pedagogy, even today (Norden, 1969).

First species counterpoint is the most restrictive species. It is often referred to as “note-against-note” counterpoint because only whole notes are allowed (Adiloglu and Alpaslan, 2007). The rules of first species counterpoint apply to a polyphonic musical fragment consisting of two parts or melodies: a cantus firmus and a counterpoint melody. The cantus firmus is a base melody to which the counterpoint melody is added. This latter melody takes into account not only the melodic transitions between notes within the melody (i.e., the horizontal aspect of the music), but also the harmonic interplay between the two melodies (i.e., the vertical aspect) (Aguilera et al., 2010). The fact that Fux’s species counterpoint can be reduced to a set of simple rules (Rothgeb, 1975) makes it easy to use them as quantifiers of quality in an optimisation context. In this research, an extensive set of first species counterpoint rules is used to test if a VNS can be used to automatically compose music. In future
research we plan to quantify rules of other musical styles that may be more relevant for modern composers, beginning with more complex counterpoint in the next chapter and composer-specific music in Chapter 4.

Contrary to existing work, the objective function developed in this chapter spans Fux’s complete theory, and is based on the qualitative Fuxian rules formalised by Salzer and Schachter (1969). Full details are given in appendix A. Some examples of rules are listed below.

- Each large leap should be followed by stepwise motion in the opposite direction.

- Only consonant (i.e., stable) intervals are allowed.

- The climax should be melodically consonant with the tonic.

- All perfect intervals should be approached by contrary or oblique motion.

In order to determine how well a generated musical fragment $s$, consisting of a cantus firmus (CF) and a counterpoint (CP) melody, adheres to the counterpoint style, the Fuxian descriptive rules are quantified. The objective functions for the CF and CP melodies are represented in equations (1.1) and (1.2) respectively. The objective function for the entire fragment is the sum of these two, as shown in equation (1.3). Both objective functions are the sum of several subscores, one per rule, whereby each subscore takes a value between 0 and 1. The lower the score, the better the fragment adheres to the rule. A “perfect” musical fragment therefore has an objective function value of 0. The relative importance of a subscore is determined by its weight. The weights $a_i$ (for the subscores of “horizontal” rules) and $b_i$ (for the subscores of “vertical” rules) can be set by the composer to calculate the total score. The score for the CF melody ($f_{CF}$) only consists of a horizontal aspect, whereas that of the CP melody ($f_{CF}$), takes into account the horizontal scores of the CP melody as well as the vertical scores, that evaluate the interaction between the two melodies. The total score of the fragment (see equation 1.3) is only used as a summary statistic as the CF and the CP melodies are optimised separately.
1.2. QUANTIFYING MUSICAL QUALITY

\[ f_{\text{CF}}(s) = \sum_{i=1}^{18} a_i \cdot \text{subscore}_i^H(s) \]  
horizontal aspect CF \hspace{1cm} (1.1)

\[ f_{\text{CP}}(s) = \sum_{i=1}^{18} a_i \cdot \text{subscore}_i^H(s) + \sum_{j=1}^{15} b_j \cdot \text{subscore}_j^V(s) \]  
horizontal aspect CP \hspace{1cm} \text{vertical aspect CP} \hspace{1cm} (1.2)

\[ f(s) = f_{\text{CF}}(s) + f_{\text{CP}}(s) \] \hspace{1cm} (1.3)

The rules used in the above objective function, can be seen as “soft” constraints. Although the algorithm developed in the next section tries to find a musical fragment that violates as few of these rules as possible (and by as little as possible), a fragment that breaks some of the rules is not necessarily “infeasible”. For pieces of arbitrary length, it is additionally not known whether it is possible to adhere to all the rules. Music theory also imposes a set of “hard rules”, that have been implemented as constraints in the optimisation problem. These rules, such as “all notes are from the tonal set” and “all notes are whole notes”, always need to be satisfied, in order to obtain a feasible fragment.

The very restrictive nature of counterpoint and the large number of rules that all need to be satisfied at the same time, make it a difficult craft for human composers to master. For several of the rules, it is not easy to see or hear if they are fulfilled. E.g., it takes an extremely trained ear or a laborious manual counting of the intervals to check whether the rule “the beginning and the end of all motion needs to be consonant” has been satisfied. However, the quantification of the counterpoint rules allows them to be objectively assessed by a computer, and allows a computer to judge whether one fragment is “better” than another. This information is used in the algorithm developed in the next section.
As an example, the score on the objective function of the counterpoint textbook example given by Salzer and Schachter (1969) (see Figure 1.1) is calculated. As expected, both the horizontal (respectively 0.44 and 0.64) and the vertical part (0.5) of the score are very good and close to the optimum ($f(s) = 0$), yet not totally optimal. This shows that even experienced composers can consider a fragment that does not obey all Fuxian rules as good counterpoint. As a comparison, the average score of 50 random fragments of the same length is 16.39.

1.3 VARIABLE NEIGHBOURHOOD SEARCH TO GENERATE COUNTERPOINT FRAGMENTS

Most of the existing research on the use of metaheuristics for CAC revolves around evolutionary algorithms. The popularity of these algorithms can partially be explained by their “black-box” character. Essentially, an evolutionary algorithm only requires that a “fitness” (objective) function can be calculated. Other components, such as the recombination operator or the selection operator, do not require any knowledge of the problem that is being solved. This makes it very easy to implement an evolutionary algorithm, but the resulting algorithm is not likely to be efficient, compared to other metaheuristics that exploit the specific structure of the problem.
Local search heuristics, on the other hand, are generally more focused on the specific problem and use far less randomness to search for good solutions. These methods typically operate by iteratively performing a series of small changes, called *moves*, on a *current solution* \( s \). The *neighbourhood* \( N(s) \) consists of all feasible solutions that can be reached in one single move from the current solution. The type of move therefore defines the neighbourhood of each solution. The local search algorithm always selects a better fragment, i.e., a solution with a better objective function value, from the neighbourhood. This solution becomes the new current solution, and the process continues until there is no better fragment in the neighbourhood. It is said that the search has arrived in a *local optimum* (see Figure 1.2) (Hansen et al., 2001).

![Figure 1.2: A perturbation move is used to “jump” out of a local optimum](image)

To escape from a local optimum, the local search metaheuristic called VNS (variable neighbourhood search) uses two mechanisms. First, instead of exploring just one neighbourhood, the algorithm switches to a different neighbourhood (defined by a different move type) if it arrives in a local optimum (Mladenovic and Hansen, 1997). The underlying idea is that a local optimum is specific to a certain neighbourhood. By allowing other move types, the search can continue. A second mechanism used by VNS is a so-called *perturbation*, that randomly changes a relatively large part of the current solution. This strategy of iteratively building a sequence of solutions leads to far
better results than randomly restarting the algorithm when it reaches a local optimum (Lourenço et al., 2003).

Although VNS is a relatively recent metaheuristic, it has been successfully applied to a broad range of optimisation problems. Davidović et al. (2005) developed a VNS for the multiprocessor scheduling problem with communication delays. This VNS outperforms both the tabu search and genetic search algorithm developed in the same chapter. Other applications of VNS include, but are not limited to, vehicle routing (Kytöjoki et al., 2007), project scheduling (Fleszar and Hindi, 2004), finding extremal graphs (Caporossi and Hansen, 2000) and graph coloring (Avanthay et al., 2003). Hansen and Mladenović (2001) applied a VNS to five well known combinatorial optimisation problems (the Travelling Salesman Problem (TSP), the p-median problem (PM), the multi-source Weber (MW) problem, the partitioning problem and the bilinear programming problem with bilinear constraints (BBLP)). Their comparison with recent heuristics showed that VNS is very efficient for large PM and MW problems. For several other problems VNS outperforms existing heuristics in an effective way, meaning that the best solutions are found in moderate computing time (Hansen and Mladenović, 2001).

The effectiveness and efficiency of VNS in solving combinatorial optimisation problems make it interesting to apply this technique to the domain of CAC. In the following sections, a VNS is described that was implemented for composing counterpoint music.

1.3.1 Components of the VNS algorithm

The VNS algorithm developed in this chapter operates in two sequential phases. First, the cantus firmus is generated, then the first species counterpoint melody. The algorithm used to generate both melodies is identical, and differs only in the objective function used. This two-phased design choice originated from the fact that a counterpoint line is usually composed on top of an existing cantus firmus line. It must be noted that, by keeping the cantus firmus fixed,
finding a suitable matching counterpoint line is highly dependent on the quality of this cantus firmus.

A “solution” in our algorithm is a musical fragment consisting of a cantus firmus and a first species counterpoint melody. The algorithm takes as its input the key and length of the fragment that is to be composed. The length is expressed as the number of notes and can be any multiple of 16. A fragment that has the correct number of notes, all of which are in the range of allowed pitches in the specified key, is called feasible. A set of 10 allowed sequential pitches is defined for the cantus firmus, starting from MIDI values 48 to 59, depending on the specified key. A different set of 10 pitches is defined for the counterpoint part, which starts 9 semitones higher. This means that for a fragment of $n$ notes, there are $10^n$ possible note combinations per voice. The functioning of the VNS algorithm ensures that no infeasible fragments are generated.

Three types of moves have been defined, they are represented in Table 1.1 and Figure 1.3.

The first neighbourhood ($N_1$) is defined by the Swap move type and is generated by swapping every pair of notes, starting from the current fragment. An example of a swap move is shown in Figure 1.3(b). $N_1$ will thus contain all possible fragments that can be obtained by swapping two notes in the current fragment.

Neighbourhoods $N_2$ and $N_3$ are respectively defined by move types Change1 and Change2. The Change1 move will change the pitch of any one note to any other allowed pitch. The last move, Change2, is an extension of the previous one whereby the pitches of two sequential notes are changed simultaneously to any other allowed pitches. These last two moves are illustrated in Figures 1.3(c) and 1.3(d).

The size of the neighbourhoods can be calculated with the formulas in Table 1.1. As an example, the relative size of the neighbourhoods for a fragment of 64 notes are 2016, 640 and 6400.
(a) Original

(b) Swap move

(c) Change1 move

(d) Change2 move

Figure 1.3.: Moves

Table 1.1.: Neighbourhoods

<table>
<thead>
<tr>
<th>$N_i$</th>
<th>Name</th>
<th>Description</th>
<th>Neighbourhood size</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_1$</td>
<td>Swap</td>
<td>Swap two notes</td>
<td>$\frac{(16 \times L)}{2}$</td>
</tr>
<tr>
<td>$N_2$</td>
<td>Change1</td>
<td>Change one note</td>
<td>$160 \times L$</td>
</tr>
<tr>
<td>$N_3$</td>
<td>Change2</td>
<td>Change two notes</td>
<td>$1600 \times L$</td>
</tr>
</tbody>
</table>

L is the length of the fragment expressed in units of 16 notes.

For each neighbourhood, $N_1$, $N_2$ and $N_3$, the algorithm generates all possible feasible fragments $s$ by applying the corresponding move and selects the one with the best value in the objective function $f(s)$. This strategy is called a steepest descent and typically ensures a fast improvement of the value of the objective function.

When no improving fragments can be found in any of the neighbourhoods of the current fragment, the VNS algorithm uses a perturbation strategy to allow the search to continue. The perturbation is implemented by reverting back to the global best fragment and changing a predefined percentage of the notes to a new, random note from the key. The reason for performing the perturbation move from the global best fragment, and not from the current fragment, is that this strategy was found to lead to better fragments faster.
Often, the current fragment scores optimally with respect to a large majority of subscores, but performs badly with respect to some others. To correct this behaviour, the VNS uses an *adaptive weight adjustment* mechanism. This mechanism adapts the weights of the subscores of the objective function at the same time a random perturbation is performed. The default initial setting for all of the weights is 1, making them normalised and all equally important in the calculation of the objective score. These initial weights can be changed by the user, in order to favour specific rules. The adaptive weight mechanism works by increasing the weight of the subscore that has the highest value (i.e., the subscore on which the current fragment has worst performance) by 1 immediately after the perturbation move. The algorithm then uses the score based on these new weights (called the *adaptive score* $f^a(s)$) to determine the quality of fragments in the neighbourhoods. In order to determine whether a fragment is considered as the new global best, however, the algorithm always considers the original weights. This weight adjustment strategy increases the impact of otherwise insignificant moves with little impact on the objective function. An increase of 1 ensures a constant pressure on subscores that are high during a particular run. In principle, the weights can become as high as the number of perturbations performed. However, since subscores with high weights are taken into account more during a move, their values generally decrease quickly, which causes the adaptive weights mechanism to favour other subscores. Because the weights never decrease, the probability that such a subscore will increase again, is very small. An increase of 1 will usually still have an impact on the newly favoured subscore, since the value of the subscores with high weights are usually very low or 0, which partly neutralises their weights even if they had become very high.

After the perturbation, the VNS always uses the *Swap* move first. The reason for this is that this neighbourhood does not introduce new notes, and therefore the search cannot immediately converge back to the previous local optimum.

To prevent the algorithm from getting trapped in cycles (i.e., revisiting the same local optimum again and again), a simple short term memory structure is introduced. This tabu list (Glover and Laguna, 1993) prohibits notes in certain places from being changed. More specifically, the notes in places that have been changed in the previous iterations by a move cannot be changed by
a move of the same type. The resulting moves are referred to as tabu active. The number of iterations that a move remains tabu active is called the tabu tenure. Each neighbourhood has its own tabu list, including its own tabu tenure. The tabu lists work by storing the recently changed notes and keeps them from being changed again by the same type of move. The tabu tenure is expressed as a fraction of the length of the melody in number of notes.

1.3.2 General outline of the implemented VNS algorithm

Figure 1.4 depicts a flow chart representation of the developed VNS algorithm. The VNS starts by generating a fragment $s$ consisting of random notes from the key. This fragment is set as the initial global best solution $s_{best}$.

For the current solution $s$, the Swap neighbourhood ($N_1$) is generated. Moves on the tabu list of $N_1$ are excluded and a feasibility check is performed. The best solution $s'$ of this neighbourhood is selected as the new current solution if its objective function with adaptive weights is better than that of $s$ ($f^a(s') < f^a(s)$). The move applied to obtain $s'$ is added to the tabu list of the Swap neighbourhood on a first in first out basis. This procedure is repeated as long as a better current solution is found, based on the objective function with adaptive weights. Each time a move is performed, the current solution $s$ is compared to the best global solution $s_{best}$, based on the original objective function $f(s)$. If $f(s)$ is better than $f(s_{best})$, the current solution $s$ becomes the new global best solution $s_{best}$.

If no better current solution, based on $f^a(s)$, is found in the first neighbourhood, the algorithm switches to the Change1 neighbourhood ($N_2$) and repeats the same procedure. If it again cannot find a better current solution in the Change1 neighbourhood, it switches to the Change2 neighbourhood ($N_3$).

If the algorithm goes through all three of the neighbourhoods without finding a better current solution (based on $f^a(s)$), then a perturbation is performed. $r\%$ of the notes of $s$ are randomly changed to form the new current solution $s$. The weight of the subscore of the best solution ($s_{best}$) that has the highest value
is also increased by one. If $f(s)$ is better than $f(s_{\text{best}})$, the current solution $s$ becomes the new global best solution $s_{\text{best}}$.

The algorithm continues to repeat these steps (generating the three types of neighbourhoods), until either the optimum is found or it has reached the maximum number of iterations without improving the best global solution $s_{\text{best}}$, as specified by the user through the $\text{maxiters}$ parameter.
CHAPTER 1. COMPOSING FIRST SPECIES COUNTERPOINT WITH VNS

1.4 IMPLEMENTATION

The VNS algorithm has been implemented in C++. To improve the usability of the software, a plug-in for the open source music composition and notation program MuseScore has been developed in JavaScript using the QtScript engine (Figure 1.5). This allows for easy interaction with Optimuse (the implementation of the VNS algorithm) through a graphical user interface. A cantus firmus can either be generated from a menu link, or composed on screen by clicking on the staff. When the cantus firmus has been generated, the counterpoint melody can be generated from a second menu link. The user can specify the input parameters such as the key (e.g., G# minor) and the weights for each subscore in an input file (input.txt). Other parameters of the algorithm, such as the ones discussed in the next section can optionally be passed as command line arguments. The resulting counterpoint music is displayed in score notation and can immediately be played back. MuseScore also provides easy export to MIDI, PDF, lilypond and other popular music notation formats. The fragment is also automatically exported in the MusicXML format, a relatively new format that is designed to facilitate music information retrieval. MuseScore can be downloaded at musescore.org. Optimuse is available for download at antor.ua.ac.be/optimuse.
1.5 EXPERIMENTS

An experiment is performed to set the parameters of the developed variable neighbourhood search algorithm to their optimal levels. The VNS is then compared to a random search and a genetic algorithm. Finally, an example of generated music is presented.

1.5.1 VNS parameter optimisation

The VNS algorithm described in Section 1.3 has several components. In this section, an experiment is described that has been set up to test the effectiveness of the different parts of the developed VNS and to determine their optimal parameter settings. In this way, components of the algorithm that do not contribute to the final solution quality can be removed. Separate experiments were performed for the generation of cantus firmus and of counterpoint. The counterpoint experiment uses the cantus firmus generated by the VNS with the same parameter settings. The different parameters that have been tested are displayed in Table 1.2. In Section 1.5.1, a small fragment is generated with the best parameter settings. The algorithm can generate musical fragments of any length, as long as the number of notes is a multiple of 16. The experiment has been limited to fragments with a length of 16, 32, 48 and 64 notes.
A full-factorial experiment was performed for each of the experimental groups (cantus firmus and counterpoint). This means that it includes some runs that have all three neighbourhoods deactivated. In that case the algorithm simply performs perturbations, depending on the random move factor. The total number of runs for both groups is \( n = 4068 \) \((2 \times 3^2 \times 4^4)\). The results of these 4068 runs of the algorithm were analysed by performing a Multi-Way ANOVA (Analysis of Variance). Using the open source statistical software R, a model was estimated that takes into account these basic variables (see Tables 1.3 and 1.4) and analyses their impact on the value of the objective function, as well as on the running time of the algorithm.

_Cantus Firmus_

Table 1.2.: Parameters of the VNS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
<th>Nr. of levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N_1 ) - Swap</td>
<td>on with ( tt_1 = 0, tt_1 = \frac{1}{4}, tt_1 = \frac{1}{4}, ) off</td>
<td>4</td>
</tr>
<tr>
<td>( N_2 ) - Change1</td>
<td>on with ( tt_2 = 0, tt_2 = \frac{1}{4}, tt_2 = \frac{1}{4}, ) off</td>
<td>4</td>
</tr>
<tr>
<td>( N_3 ) - Change2</td>
<td>on with ( tt_3 = 0, tt_3 = \frac{1}{4}, tt_3 = \frac{1}{4}, ) off</td>
<td>4</td>
</tr>
<tr>
<td>Random move (randsize)</td>
<td>( \frac{1}{4} ) changed, ( \frac{1}{8} ) changed, off</td>
<td>3</td>
</tr>
<tr>
<td>Adaptive weights (adj. weights)</td>
<td>on, off</td>
<td>2</td>
</tr>
<tr>
<td>Max. number of iterations (iters)</td>
<td>10, 50, 100</td>
<td>3</td>
</tr>
<tr>
<td>Length of music (length)</td>
<td>16, 32, 48, 64 notes</td>
<td>4</td>
</tr>
</tbody>
</table>

\( tt_i \) = tabu tenure of the tabu list of neighbourhood \( N_i \), expressed as a fraction of the total number of notes.

Table 1.3.: Model without interactions (CF) - Summary of Fit

<table>
<thead>
<tr>
<th>Measure</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R^2 )</td>
<td>0.2605</td>
</tr>
<tr>
<td>( R^2 ) Adj</td>
<td>0.2577</td>
</tr>
<tr>
<td>F-statistic</td>
<td>95.1</td>
</tr>
<tr>
<td>p-value</td>
<td>(&lt; 2.2e^{-16})</td>
</tr>
</tbody>
</table>
When including only the main effects, the $R^2$ statistic of the linear regression (Table 1.3) shows that approximately 26% of the total variation around the mean value of the objective function can be explained by the model. However, the value of the $R^2$ statistic and therefore also the quality of the model can be increased by including interaction effects. In order to determine which parameters have a significant influence on the quality of the generated music, a model was calculated that takes into account the interaction effects of the significant factors ($p < 0.05$). This high quality model has an $R^2$ statistic of approximately 96% (Table 1.5), which means that it can explain 96% of the variation. The most important influential factors of the model are displayed in Table 1.6.

When including only the main effects, the $R^2$ statistic of the linear regression (Table 1.3) shows that approximately 26% of the total variation around the mean value of the objective function can be explained by the model. However, the value of the $R^2$ statistic and therefore also the quality of the model can be increased by including interaction effects. In order to determine which parameters have a significant influence on the quality of the generated music, a model was calculated that takes into account the interaction effects of the significant factors ($p < 0.05$). This high quality model has an $R^2$ statistic of approximately 96% (Table 1.5), which means that it can explain 96% of the variation. The most important influential factors of the model are displayed in Table 1.6.

### Table 1.4.: Multi-Way ANOVA model without interactions (CF)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Df</th>
<th>$F$ value</th>
<th>Prob (&gt;F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_1$</td>
<td>1</td>
<td>220.3070</td>
<td>$&lt; 2.2e^{-16} , \ast$</td>
</tr>
<tr>
<td>$N_2$</td>
<td>1</td>
<td>369.0956</td>
<td>$&lt; 2.2e^{-16} , \ast$</td>
</tr>
<tr>
<td>$N_3$</td>
<td>1</td>
<td>731.9956</td>
<td>$9.702e^{-06} , \ast$</td>
</tr>
<tr>
<td>$tt_1$</td>
<td>2</td>
<td>0.0373</td>
<td>0.9634</td>
</tr>
<tr>
<td>$tt_2$</td>
<td>2</td>
<td>0.6274</td>
<td>0.5340</td>
</tr>
<tr>
<td>$tt_3$</td>
<td>2</td>
<td>0.6477</td>
<td>0.5233</td>
</tr>
<tr>
<td>randsize</td>
<td>2</td>
<td>111.4955</td>
<td>$&lt; 2.2e^{-16} , \ast$</td>
</tr>
<tr>
<td>adj. weights</td>
<td>1</td>
<td>19.6100</td>
<td>$9.718e^{-06} , \ast$</td>
</tr>
<tr>
<td>iters</td>
<td>2</td>
<td>14.1792</td>
<td>$7.261e^{-07} , \ast$</td>
</tr>
<tr>
<td>length</td>
<td>3</td>
<td>7.2613</td>
<td>$7.406e^{-05} , \ast$</td>
</tr>
</tbody>
</table>

### Table 1.5.: Model with interactions (CF) - Summary of Fit

<table>
<thead>
<tr>
<th>Measure</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R^2$</td>
<td>0.9612</td>
</tr>
<tr>
<td>$R^2$ Adj</td>
<td>0.9556</td>
</tr>
<tr>
<td>$F$-statistic</td>
<td>171.8</td>
</tr>
<tr>
<td>$p$-value</td>
<td>$&lt; 2.2e^{-16}$</td>
</tr>
</tbody>
</table>
A similar Multi-Way ANOVA test has been performed, using the computation time of the algorithm as the dependent variable. The results of this analysis are displayed in Table 1.7 and show that all factors, except for the tabu list tenure for \(N_1\) and \(N_3\), have a significant influence on the running time.

Examination of the results of the ANOVA Table with interaction effects reveals that most of the factors have a very low \(p\)-value. This means that they have a significant influence on the result. The large number of small \(p\)-values indicates that most of the factors make a contribution to the model. Only \(tt_1\), the tabu tenure of \(N_1\) does not seem to have a significant influence on the quality of the result. The exact effect that the different parameters settings have on the objective function of the end result is visualised in their means plots (Figure 1.6).

The ANOVA Table (Table 1.6) shows that parameters \(N_1\), \(N_2\) and \(N_3\) all have a significant influence on the quality of the generated music (\(p\)-value < \(2e^{-16}\)). Figures 1.6(a), 1.6(b) and 1.6(c) show that activating one of the neighbourhoods will have a decreasing effect on the score. However, they also show that adding a neighbourhood increases the mean running time for both \(N_1\) and \(N_3\). According to Table 1.7 this effect on the running time is significant. In deciding the optimal parameter settings, the primary objective was the musical quality; the computing time was considered a secondary objective. The three neighbourhoods will therefore be included in the optimal parameter set.

From Tables 1.7 and 1.6, it is clear that the tabu tenure for \(N_1\) does not have a significant effect on the quality or the computing time (\(p\)-values 0.5 and 0.7). The tabu tenures for \(N_2\) and \(N_3\) do have a significant influence. The means plots 1.7(e) and 1.7(f) show that tabu tenures of respectively \(\frac{1}{4}\) and \(\frac{1}{2}\) (i.e., 25\% and 50\% of the length of the number of notes of the melody respectively) contribute most to a higher result quality.

With \(p < 2e^{-16}\), the random perturbation size clearly has a significant effect on the solution quality. Plot 1.6(g) indicates that a perturbation will result in music of better quality, and that a perturbation of 12.5\% offers the best results in the objective score, although this has a significant negative effect on the computing time.
Table 1.6: Multi-Way ANOVA model with interactions (CF) (extract)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Df</th>
<th>F value</th>
<th>Prob (&gt; F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_1$</td>
<td></td>
<td>3.685e+07</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>$N_2$</td>
<td></td>
<td>6.175e+07</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>$N_3$</td>
<td></td>
<td>1.224e+07</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>$t_{01}$</td>
<td></td>
<td>0.6242</td>
<td>0.535733</td>
</tr>
<tr>
<td>$t_{02}$</td>
<td></td>
<td>1.04974</td>
<td>2.837e-05  *</td>
</tr>
<tr>
<td>$t_{03}$</td>
<td></td>
<td>1.08367</td>
<td>2.025e-05  *</td>
</tr>
<tr>
<td>randsize</td>
<td></td>
<td>1.865e+07</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>adj. weights</td>
<td></td>
<td>3.28102</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>iters</td>
<td></td>
<td>2.337231</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>length</td>
<td></td>
<td>1.214919</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>$N_1$:N_2</td>
<td></td>
<td>7.808e+07</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>$N_1$:N_3</td>
<td></td>
<td>9.565e+07</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>$N_2$:N_3</td>
<td></td>
<td>1.8704e+07</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>$N_1$:randsize</td>
<td></td>
<td>3.93468</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>$N_2$:randsize</td>
<td></td>
<td>2.529978</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>$N_3$:randsize</td>
<td></td>
<td>8.720523</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>$N_1$:adj. weights</td>
<td></td>
<td>1.6866</td>
<td>0.1941202</td>
</tr>
<tr>
<td>$N_2$:adj. weights</td>
<td></td>
<td>2.4398</td>
<td>0.1183666</td>
</tr>
<tr>
<td>$N_3$:adj. weights</td>
<td></td>
<td>4.5202</td>
<td>0.033557 *</td>
</tr>
<tr>
<td>randsize:adj. weights</td>
<td></td>
<td>2.2749924</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>$N_1$:iters</td>
<td></td>
<td>4.14913</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>$N_2$:iters</td>
<td></td>
<td>1.08369</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>$N_3$:iters</td>
<td></td>
<td>2.124846</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>randsize:iters</td>
<td></td>
<td>8.8365</td>
<td>4.225e-07 *</td>
</tr>
<tr>
<td>adj. weights:iters</td>
<td></td>
<td>6.0082</td>
<td>0.0024806 *</td>
</tr>
<tr>
<td>$N_1$:length</td>
<td></td>
<td>4.54746</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>$N_2$:length</td>
<td></td>
<td>6.4358</td>
<td>0.0002411</td>
</tr>
<tr>
<td>$N_3$:length</td>
<td></td>
<td>3.13985</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>randsize:length</td>
<td></td>
<td>2.7547</td>
<td>0.0112953</td>
</tr>
<tr>
<td>adj. weights:length</td>
<td></td>
<td>0.5819</td>
<td>0.6268833</td>
</tr>
<tr>
<td>iters:length</td>
<td></td>
<td>0.3941</td>
<td>0.882212</td>
</tr>
<tr>
<td>$N_1$:N_2:N_3</td>
<td></td>
<td>2.3470e+07</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>commented this for page length $N_1$:N_2:randsize</td>
<td></td>
<td>1.248853</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>$N_1$:N_2:randsize</td>
<td></td>
<td>1.235017</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>$N_2$:N_3:randsize</td>
<td></td>
<td>8.177573</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>$N_1$:N_2:adj. weights</td>
<td></td>
<td>2.4761</td>
<td>0.1156635</td>
</tr>
<tr>
<td>$N_1$:N_3:adj. weights</td>
<td></td>
<td>0.0013</td>
<td>0.9706971</td>
</tr>
<tr>
<td>$N_2$:N_3:adj. weights</td>
<td></td>
<td>0.1372</td>
<td>0.711064</td>
</tr>
<tr>
<td>$N_1$:randsize:adj. weights</td>
<td></td>
<td>6.4709</td>
<td>0.001564 *</td>
</tr>
<tr>
<td>$N_2$:randsize:adj. weights</td>
<td></td>
<td>0.2066</td>
<td>0.8133344</td>
</tr>
<tr>
<td>$N_3$:randsize:adj. weights</td>
<td></td>
<td>1.0064</td>
<td>4.367e-05  *</td>
</tr>
<tr>
<td>$N_1$:N_2:iters</td>
<td></td>
<td>0.996471</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>$N_1$:N_3:iters</td>
<td></td>
<td>9.46976</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>$N_2$:N_3:iters</td>
<td></td>
<td>3.457817</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>$N_1$:randsize:iters</td>
<td></td>
<td>6.0681</td>
<td>6.875e-05 *</td>
</tr>
<tr>
<td>$N_2$:randsize:iters</td>
<td></td>
<td>2.7069</td>
<td>0.0287158 *</td>
</tr>
<tr>
<td>$N_3$:randsize:iters</td>
<td></td>
<td>7.4022</td>
<td>6.164e-06 *</td>
</tr>
<tr>
<td>$N_1$:adj. weights:iters</td>
<td></td>
<td>0.25</td>
<td>0.7797971</td>
</tr>
<tr>
<td>$N_2$:adj. weights:iters</td>
<td></td>
<td>0.4876</td>
<td>0.614123</td>
</tr>
<tr>
<td>$N_3$:adj. weights:iters</td>
<td></td>
<td>3.1014</td>
<td>0.0451376 *</td>
</tr>
<tr>
<td>randsize:adj. weights:iters</td>
<td></td>
<td>5.3779</td>
<td>0.0002667 *</td>
</tr>
</tbody>
</table>

...
Figure 1.6.: Means plots CF
1.5. Experiments

Figure 1.7.: Mean plots CP
Table 1.7: Multi-Way ANOVA model for Time

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Df</th>
<th>$F$ value</th>
<th>Prob ($&gt; F$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_1$</td>
<td>1</td>
<td>36.6801</td>
<td>$1.503e^{-9}$*</td>
</tr>
<tr>
<td>$N_2$</td>
<td>1</td>
<td>13.3081</td>
<td>$0.0002672$*</td>
</tr>
<tr>
<td>$N_3$</td>
<td>1</td>
<td>22.4816</td>
<td>$&lt; 2.186e^{-6}$*</td>
</tr>
<tr>
<td>$tt_1$</td>
<td>2</td>
<td>0.3025</td>
<td>0.7389918</td>
</tr>
<tr>
<td>$tt_2$</td>
<td>2</td>
<td>3.8164</td>
<td>$0.0220770$*</td>
</tr>
<tr>
<td>$tt_3$</td>
<td>2</td>
<td>1.5815</td>
<td>$0.2057780$*</td>
</tr>
<tr>
<td>randsize</td>
<td>2</td>
<td>218.6638</td>
<td>$&lt; 2.2e^{-16}$*</td>
</tr>
<tr>
<td>adj. weights</td>
<td>1</td>
<td>18.2537</td>
<td>$1.973e^{-5}$*</td>
</tr>
<tr>
<td>iters</td>
<td>2</td>
<td>210.8771</td>
<td>$&lt; 2.2e^{-16}$*</td>
</tr>
<tr>
<td>length</td>
<td>3</td>
<td>691.4822</td>
<td>$&lt; 2.2e^{-16}$*</td>
</tr>
</tbody>
</table>

$R^2=0.3981$

Table 1.6 demonstrates that the adaptive weights parameter also has a significant influence on the musical quality if we take the interaction effects into account. Figure 1.6(h) shows that activating this parameter has a significant lowering effect on the mean computing time as well as the objective function. This means that the adaptive weights procedure makes a positive contribution to the general effectiveness of the VNS.

The maximum number of iterations is a significant factor in the algorithm ($p < 2.2e^{-16}$). The means plot 1.6(i) clearly shows that solution quality improves when the maximum number of iterations is higher. However, as expected, this is paired with a significant increase in computing time.

**Counterpoint**

The same full factorial experiment was run on the VNS algorithm to generate the counterpoint melody.

Table 1.8 indicates that the model with interactions can explain approximately 91% of the total variation around the mean value of the objective function.
The detailed results of this model are displayed in the ANOVA Table with interactions (Table 1.9). The ANOVA model shows the same significant parameters as the analysis in the previous section. The mean plots for the cantus firmus (Figure 1.7) also show a strong resemblance to the ones with the counterpoint results (Figure 1.6). The conclusions of the previous section can therefore be extended to the counterpoint results.

### Table 1.8.: Model with interaction effects (CP) - Summary of Fit

<table>
<thead>
<tr>
<th>Measure</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R^2$</td>
<td>0.9122</td>
</tr>
<tr>
<td>$R^2$ Adj</td>
<td>0.9062</td>
</tr>
<tr>
<td>F-statistic</td>
<td>152.4</td>
</tr>
<tr>
<td>$p$-value</td>
<td>$&lt; 2.2e^{-16}$</td>
</tr>
</tbody>
</table>

### Table 1.9.: Multi-Way ANOVA model with interaction effects (CP)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Df</th>
<th>$F$ value</th>
<th>Prob ($&gt; F$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_1$</td>
<td>1</td>
<td>1173.4292</td>
<td>$&lt; 2.2e^{-16}$ *</td>
</tr>
<tr>
<td>$N_2$</td>
<td>1</td>
<td>2618.9755</td>
<td>$&lt; 2.2e^{-16}$ *</td>
</tr>
<tr>
<td>$N_3$</td>
<td>1</td>
<td>6498.1957</td>
<td>$&lt; 2.2e^{-16}$ *</td>
</tr>
<tr>
<td>randsize</td>
<td>2</td>
<td>2610.1349</td>
<td>$&lt; 2.2e^{-16}$ *</td>
</tr>
<tr>
<td>iters</td>
<td>2</td>
<td>111.7095</td>
<td>$&lt; 2.2e^{-16}$ *</td>
</tr>
<tr>
<td>length</td>
<td>3</td>
<td>125.8989</td>
<td>$&lt; 2.2e^{-16}$ *</td>
</tr>
<tr>
<td>$tt_1$</td>
<td>2</td>
<td>1.3815</td>
<td>0.2513093 *</td>
</tr>
<tr>
<td>$tt_2$</td>
<td>2</td>
<td>7.5519</td>
<td>0.0005321 *</td>
</tr>
<tr>
<td>$tt_3$</td>
<td>2</td>
<td>188.5756</td>
<td>$&lt; 2.2e^{-16}$ *</td>
</tr>
<tr>
<td>adj. weights</td>
<td>1</td>
<td>18.5697</td>
<td>1.675e^{-05} *</td>
</tr>
</tbody>
</table>

Interactions of all significant factors are included in the model, but omitted in the table for clarity.
Optimal parameter settings

The means plots and ANOVA analysis give a good indication of the significant parameters and their optimal setting. The discrete points of the mean plots are connected in order to make them better readable. Interaction plots for the significant interaction effects, drawn in R, support the conclusions made in the previous section. Table 1.10 summarises the optimal parameter settings. As mentioned, these parameters were set in a way that always favoured solution quality over computing time. In other words, the effect of a parameter on the computing time was only taken into account if the means plot did not show a significant effect on the quality.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_1$ - Swap</td>
<td>on with $tt_1=0$</td>
</tr>
<tr>
<td>$N_2$ - Change1</td>
<td>on with $tt_2=\frac{1}{4}$</td>
</tr>
<tr>
<td>$N_3$ - Change2</td>
<td>on with $tt_3=\frac{1}{2}$</td>
</tr>
<tr>
<td>Random move</td>
<td>$\frac{1}{8}$ changed</td>
</tr>
<tr>
<td>Adaptive weights</td>
<td>on</td>
</tr>
<tr>
<td>Max. number of iterations</td>
<td>100</td>
</tr>
<tr>
<td>Length of music</td>
<td>64 notes</td>
</tr>
</tbody>
</table>

$tt_i =$ tabu tenure of the tabu list of neighbourhood $N_i$, expressed as a fraction of the total number of notes.

The algorithm was run again with these optimal parameter settings. Figure 1.8 shows the evolution of the solution quality over time, which is characterised by a fast improvement of the best solution in the beginning of the algorithm’s run. After several iterations, the improvements diminish in size, especially in the case of counterpoint. The random perturbations can be spotted on the graph as peaks in the objective score. They are usually followed by a steep descent which often results in a better “best solution”, confirming the importance of the random perturbation factor. The perturbation is typically
performed after calculating 50 to 100 neighbourhoods. This number is not exact, it varies with each run and decreases after a while.

![Graph](image)

Figure 1.8.: Evolution of the VNS with optimal parameter settings
The random initial fragment serves as a starting point for the VNS. In order to examine the influence of the initial fragment on the generated music, the VNS was run two times for a cantus firmus of 64 notes with maxiters set to 10. In the end results for both runs, only 12% and 34% of the notes of the initial random fragment were unchanged. This experiment shows that the initial fragment is generally changed quite thoroughly by the algorithm. When comparing the two generated fragments, 77% of the notes in the cantus firmus are different. Similar results are seen with the counterpoint melody. When the VNS was run twice to generate a CP, there was a 70% difference between two generated melodies and 7% and 14% resemblance with the original random fragment. This shows that two musical fragments generated by the VNS and based on the same initial fragment have very little resemblance.

1.5.2 VNS versus random search

In order to determine the efficiency of the VNS algorithm, it was compared to a random search. This comparison was performed on a musical piece consisting of 64 measures. The VNS was run for one iteration, until it reached its first local optimum. No perturbation was performed. In one iteration it performed 28 moves in $N_1$, 13 in $N_2$ and 13 in $N_3$ for the cantus firmus and 32 moves in $N_1$, 15 in $N_2$ and 12 in $N_3$ for the counterpoint. In order to evaluate the generated neighbourhoods, the algorithm had to calculate the objective score a total of respectively 147,968 and 150,912 times for cantus firmus and counterpoint. A random search generated an equal amount of solutions. The counterpoint experiment, for both VNS and random search, started from the same initial cantus firmus that was generated by the VNS with optimum parameters. Figure 1.9 shows that the VNS is clearly able to find a much better solution, especially for the counterpoint melody. Although the VNS moves slower in the very beginning of the run, it is able to find a better solution almost immediately. This conclusion is supported by Figure 1.10, where the best found objective score is plotted against the running time of the algorithms.
Figure 1.9.: Comparison of VNS and random search
1.5.3 VNS versus GA

A formal comparison with a previously developed genetic algorithm is not possible because the cited algorithms use a different fitness function than the developed VNS. In order to compare the effectiveness of the VNS with a GA, a genetic algorithm was implemented that uses the objective function described in this chapter as fitness function.

The genetic algorithm starts from a population of randomly generated solutions. Binary tournament selection is used to select two parents from the population. A two-point-crossover is performed on two random points to create the children. A mutation, changing \( r \% \) of the notes randomly, is performed on one of the children with a probability \( p \). The children are then reinserted in the population with reversed binary tournament selection. A child is excluded from the population if an equal population member already exists. This keeps the population from converging.

In the developed GA, a number of parameters can be set to different levels. A similar full factorial experiment to the one performed on the VNS was done to determine the best settings for the parameters represented in Table 1.11. The GA was run with each of the settings for the cantus firmus and the counterpoint. This resulted in 864 runs. The counterpoint takes the cantus firmus generated by the GA with the same parameter settings as input.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
<th>Nr. of levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>10, 100, 1000</td>
<td>3</td>
</tr>
<tr>
<td>Mutation size</td>
<td>0%, 12.5%, 25%</td>
<td>3</td>
</tr>
<tr>
<td>Mutation frequency</td>
<td>5%, 10%</td>
<td>2</td>
</tr>
<tr>
<td>Number of generations (gens)</td>
<td>150,000, 750,000, 1,500,000</td>
<td>3</td>
</tr>
<tr>
<td>Length of music (length)</td>
<td>16, 32, 48, 64 notes</td>
<td>4</td>
</tr>
</tbody>
</table>
An ANOVA model was constructed from the results. The initial model, both for CF and CP, shows that almost all factors are significant. Only the number of generations and the mutation frequency do not have a significant influence on the solution quality. A new model with interaction effects between significant factors was built. The results are similar for the cantus firmus and the counterpoint, except that the number of generations is not significant for the CF. The ANOVA summary for the CP is displayed in Table 1.12. The optimal parameter settings with regards to the solution quality are displayed in Table 1.13. These were obtained by an analysis of the mean plots and interaction plots outputted by R.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Df</th>
<th>F value</th>
<th>Prob (&gt;F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>2</td>
<td>307.632</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>Mutation size</td>
<td>2</td>
<td>1114.9281</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>Length</td>
<td>3</td>
<td>193.1132</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>Mutation frequency</td>
<td>1</td>
<td>2.3477</td>
<td>0.127255</td>
</tr>
<tr>
<td>Gens</td>
<td>2</td>
<td>10.8473</td>
<td>3.599e-05 *</td>
</tr>
<tr>
<td>Population size:Mutation size</td>
<td>4</td>
<td>362.2764</td>
<td>&lt; 2.2e-16 *</td>
</tr>
<tr>
<td>Population size:Length</td>
<td>6</td>
<td>3.0996</td>
<td>0.006535 *</td>
</tr>
<tr>
<td>Mutation size:Length</td>
<td>6</td>
<td>8.9242</td>
<td>1.634e-08 *</td>
</tr>
<tr>
<td>Population size:Mutation size:Length</td>
<td>12</td>
<td>1.0715</td>
<td>0.386779</td>
</tr>
</tbody>
</table>

$R^2=0.9657$

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>1000</td>
</tr>
<tr>
<td>Mutation size</td>
<td>12.5%</td>
</tr>
<tr>
<td>Mutation frequency</td>
<td>10%</td>
</tr>
<tr>
<td>Number of generations</td>
<td>1,500,000</td>
</tr>
</tbody>
</table>

In order to compare the efficiency of both algorithms another experiment was set up. The VNS and GA algorithms both generated 50 cantus firmus and counterpoint fragments of 64 notes. The 50 cantus firmus instances generated
by the VNS were used by both the GA and the VNS as input to compose a counterpoint melody. The algorithms used the optimal parameter settings that resulted from the above analysis. The cut-off for the VNS is 25 maxiters, and for the GA respectively 1,500,000 and 3,000,000 generations for cantus firmus and counterpoint. This resulted in comparable running times for both algorithms, although the average running time of the genetic algorithm is slightly longer than that of the VNS (see Table 1.14). The computer used in this experiment has an Intel\textsuperscript{®} Core\textsuperscript{TM} i7 CPU which runs at 2.93GHz. To ensure a fair comparison, both algorithms are coded in C++ and large parts of their code-base is the same. For example, both algorithms use the same code to calculate the objective function.

<table>
<thead>
<tr>
<th></th>
<th>Mean running time VNS</th>
<th>Mean running time GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>CF</td>
<td>563s</td>
<td>795s</td>
</tr>
<tr>
<td>CP</td>
<td>1823s</td>
<td>1992s</td>
</tr>
</tbody>
</table>

The findings of this experiment were analysed by a unilateral paired t-test. The resulting p-values of 0.01593 for the cantus firmus and $5.363\times10^{-15}$ for the counterpoint show that the VNS is able to find a better solution than the GA, despite its shorter mean running time.

Figure 1.10 shows the evolution over time of the best found solution for the VNS, GA and random search. When generating the cantus firmus, all algorithms start from a randomly generated solution. For the generation of the counterpoint fragment all three runs are based on the same cantus firmus generated by the VNS with the optimal settings. The graph shows that although the genetic algorithm finds slightly better solutions in the beginning of the run, the VNS finds a better solution than the GA almost immediately. The GA does not converge, since duplicate population members are not allowed. This is confirmed by the fact that the algorithm is still able to make small improvements to the quality of the best solution long into the GA run, for example from 2.83698 to 2.8234 after 984 seconds. In Figure 1.9 the best found solution of the algorithms is plotted against the number of times a solution is evaluated. This graph confirms the previous conclusions. Overall,
Figure 1.10: Comparison of VNS and GA
the GA and the Random search are completely outperformed by the VNS, since none of them comes close to finding the best solution found by the VNS.

1.6 Conclusions

In this chapter, an efficient VNS algorithm has been developed to automatically “compose” musical fragments consisting of a cantus firmus and a first species counterpoint melody. To this end the first species counterpoint rules have been quantified and used as an objective function in a local search algorithm. The different parameter settings of the VNS were extensively analysed by means of a full factorial experiment, which resulted in a set of optimal parameter settings. A comparison with a random search and a genetic algorithm confirmed its efficiency. The resulting algorithm was then implemented in a user-friendly way. The musical output of the VNS has a good objective score. The fragments could, even at this point, be used by a composer as a starting point for a musical composition.

Figure 1.11.: Generated counterpoint with score of 0.371394
An example of the music that is generated by the VNS with the optimal settings is displayed in Figure 1.11. With an objective score as low as 0.371394, this fragment is very close to violating none of the Fuxian counterpoint rules. Although a this is a subjective interpretation, the fragment is pleasant to the ear and sounds a lot less “random” than the fragment generated as the initial solution. This and other demo pieces are available for download at antor.ua.ac.be/optimuse.

The downside to using these highly restrictive rules is that they are also very limiting. In addition to the obvious limitation of using only whole notes, there are no rules enforcing a theme or coherence in the music, which causes a sense of meandering, especially in longer fragments.

An interesting extension of this work would be to evaluate different styles and types of music. A rhythmic component can be added to the music, by working with other species of counterpoint, such as florid counterpoint (see Chapter 2). The number of parts can also be increased, to allow more voices at the same time. It might also be interesting to evaluate music from a larger perspective and add a sense of direction or theme. Generating music into a structure will be explored in Chapter 6.

Another possible extension of the existing objective function is to add composer specific characteristics. Manaris et al. (2005) developed a set of 10 composer specific metrics by scanning musical databases. These metrics, all based on Zipf’s law, include the frequency distribution of pitch, duration, harmonic and melodic intervals. An artificial neural network was used to classify pieces in terms of authorship and style. While they are adequate for composer classification, these criteria alone do not seem to be sufficient for generating aesthetically pleasing music (Manaris et al., 2003). A combination of similar composer specific simple metrics, with the objective function developed in this chapter might offer an interesting approach to evaluate composer specific classical music. This is explored in Chapter 4.

In the next chapter, the developed metaheuristic is adapted to generate fifth species counterpoint.
COMPOSING FIFTH SPECIES COUNTERPOINT MUSIC WITH A VARIABLE NEIGHBORHOOD SEARCH ALGORITHM

This chapter is based on the paper “D. Herremans, K. Sörensen. 2013. Composing Fifth Species Counterpoint Music With A Variable Neighborhood Search Algorithm. Expert Systems with Applications. 40(16).”
In this chapter, the variable neighbourhood search algorithm developed in the previous chapter is expanded to work with fifth species counterpoint instead of first species, a more complex form of counterpoint that also includes a rhythmic aspect. The existing fifth species counterpoint rules are quantified and form the basis of the objective function used by the algorithm. The VNS implemented in this research is a local search metaheuristic that starts from a randomly generated fragment and gradually improves this solution by changing one or two notes at a time. An in-depth statistical analysis reveals the significance as well as the optimal settings of the parameters of the VNS. The algorithm has been implemented in a user-friendly software environment called Optimuse. Optimuse allows a user to input basic characteristics such as length, key and mode. Based on this input, a fifth species counterpoint fragment is generated by the system that can be edited and played back immediately. The structure of this chapter is similar to that of the previous one. Part of the literature review of the original paper has been moved to the previous chapter. Yet, there might still be a slight overlap which has been kept in order to preserve the structure and logical connections within this chapter.

2.1 INTRODUCTION

From the very conception of computers, the idea was formed that they could be used as a tool for composers. Around 1840, Ada Lovelace, the world’s first conceptual programmer (Gürer, 2002) hinted at using computers for automated composition:

“[The Engine’s] operating mechanism might act upon other things besides numbers [. . .] Supposing, for instance, that the fundamental relations of pitched sounds in the signs of harmony and of musical composition were susceptible of such expressions and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent.” – (Bowles, 1970)

Starting in the mid 1900s, many systems have been developed for automated composition, both for melody harmonization (i.e., finding the most musically suitable accompaniment to a given melody) (Raczyński et al., 2013), generating
a melodic line to a given chord sequence or cantus firmus \citep{Herremans2012} and even generating a full musical piece from scratch \citep{Sandred2009}. A more complete summary of these systems is given in Section 1.1.

In this chapter, the first successful implementation of a variable neighbourhood search algorithm (VNS) to generate first species counterpoint (see Chapter 1) is expanded to fifth species counterpoint music. This algorithm was implemented as a tool called Optimuse. In this research, the existing work is expanded to generate more complex fifth species counterpoint.

The next section gives an overview of existing research and explains the difference between first and fifth species counterpoint. Contrary to most of the existing studies, a detailed breakdown of the objective function is given and Fux’s rules are included as extensively as possible. In Section 2.3 the developed objective function that assesses how well music fits into the counterpoint style is explained. This is followed by a detailed description of the algorithm in Section 2.4. Details on how Optimuse was implemented are discussed in Section 2.5. In Section 2.6, a statistical experiment is described that determines the optimal parameter settings of the VNS, this is followed by some general conclusions.

2.2 FROM FIRST TO FIFTH SPECIES

In Chapter 1 a variable neighbourhood search algorithm was implemented that can generate cantus firmi and first species counterpoint melodies. Figure 2.1 shows an example of a first species counterpoint fragment. In this figure, the bottom line is the cantus firmus (CF), or “fixed song”. The top line is the counterpoint (CP), a melody that is composed by not only taking into account the melodic relationship between the subsequent notes, but also the harmonic balance with the cantus firmus.

Since the previously developed algorithm was successful in generating good cantus firmi and first species counterpoint fragments, Optimuse was extended to include fifth species, a more complex form of counterpoint music. In fifth
species, a rhythmical aspect is added to the music. An example of fifth species counterpoint is displayed in Figure 2.2.

Figure 2.1.: First species counterpoint fragment (generated by Optimuse)

Figure 2.2.: Fifth species counterpoint fragment (Salzer and Schachter, 1969)

The amount of research done on automatically evaluating a musical fragment, to circumvent the human bottleneck, is very limited. A musical style is not typically defined in a formal way that can be quantified. However, the counterpoint style is an exception to this rule. It is a very formalized and restrictive style with simple rules for melody and harmony (Rothgeb, 1975), which makes it perfect for quantification.

A number of studies use counterpoint rules to evaluate the generated music. However, these often implement only a small subset of Fux’s rules. Aguilera et al. (2010) developed an algorithm that uses probabilistic logic to generate first species counterpoint music in C major. Their fitness function uses only the harmonic characteristics of counterpoint and ignores the melodic aspects. Composer David Cope’s “Gradus”, composes first species counterpoint given a cantus firmus. The evaluation is based on six criteria, a small subset of all counterpoint rules (Cope, 2004). The GA developed by Phon-Amnuaisuk et al.
(1999) uses a set of four-part harmonization rules as a fitness function. Donnelly and Sheppard (2011) presented a similar GA, based on 10 melodic, 3 harmonic and 2 rhythmic criteria. For a more extensive overview of existing research, the reader is referred to Chapter 1.

This research includes Fux’s rules as extensively as possible. In the next section, a detailed breakdown of the objective function is given.

2.3 QUANTIFYING MUSICAL QUALITY

In this chapter, Optimuse is expanded to generate fifth species counterpoint music, a very specific type of polyphonic classical music. Counterpoint music was the inspiration of many of the great composers such as Bach and Haydn and is foundational in music pedagogy, even today (Mateos-Moreno, 2011). The counterpoint rules are a set of strict and specific rules that take into account the complexities that arise by playing multiple notes at the same time (Siddharthan, 1999). Johann Fux wrote down very specific counterpoint rules in his Gradus Ad Parnassum in 1725, a pedagogical book designed to teach musical students how to compose (Fux and Mann, 1971). It starts by explaining rules for easy (first species) musical fragments and gradually moves to more complex (fifth species) music. Each of Fux’s species can be seen as “levels” that add more complexity to the music, e.g. more rhythmical possibilities (Adiloglu and Alpaslan, 2007).

Just like a student would start with Fux’s first species, the cantus firmus and first species rules were implemented in Optimuse as a prototype in the previous chapter. Each of the rules, as described by Salzer and Schachter (1969), were quantified into a subscore between 0 and 1. These subscores include rules such as “each large leap should be followed by stepwise motion in the opposite direction” and “the climax should be melodically consonant with the tonic”. This chapter brings Optimuse to the next level by extending the rules of its objective function with fifth species counterpoint rules. The subscores now also include rules for more complex musical structures such
as passing notes, ties, ottava and quinta battuta. A passing note is a non-harmonic note that appears between two stepwise moving notes. An ottava or quinta battuta (beaten octave or fifth) occurs when two voices move in contrary motion and leap into a perfect consonance (Salzer and Schachter, 1969). The rules can be divided into two categories. Melodic rules focus on the horizontal relationship between successive notes, whereas harmonic rules focus on the vertical interplay between simultaneously sounding notes. A detailed breakdown of the resulting subscores can be found in Appendix B.

Given a cantus firmus and a fifth species counterpoint fragment, the objective function \( f(s) \), displayed in Equation 2.1, calculates how good the fragment fits into the counterpoint style.

\[
f(s) = \sum_{i=1}^{19} a_i \cdot \text{subscore}_i^H(s) + \sum_{j=1}^{19} b_j \cdot \text{subscore}_j^V(s)
\]

This score is used as an indicator of quality of the generated music. All subscores result in a number between 0 (best) and 1 (worst), therefore, the objective of the algorithm developed in the next section is to minimize \( f(s) \). Each subscore has a weight \( a_i \) or \( b_j \) and the total score is a linear combination of the subscores with their corresponding weights. The weights are set in the beginning by the user. This allows a particular rule to be emphasized according to the preferences of the user.

The rules mentioned above are all “soft” rules. Although the aim of Optimuse is to find a musical fragment that has the lowest possible value for the objective function, it is allowed that a few of these rules are “broken”, meaning that their corresponding subscore is not equal to zero. Given the large number of rules and their complexity, it not known if it is even possible to satisfy all of them at the same time for a piece of arbitrary length.

The soft rules are supplemented with a set of “hard” rules, that have been implemented as constraints. While soft rules can be violated, hard rules cannot. A violation of one or more of these constraints renders the musical fragment
2.4. Variable Neighbourhood Search

Infeasible. All rhythmic criteria that are defined by Salzer and Schachter (1969) have been implemented as hard constraints. Table 2.1 lists the implemented feasibility criteria.

<table>
<thead>
<tr>
<th>No</th>
<th>Feasibility criterium</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>All notes come from the correct key.</td>
</tr>
<tr>
<td>2</td>
<td>Only certain rhythmic patterns are allowed for a measure.</td>
</tr>
<tr>
<td>3</td>
<td>No rhythmic pattern can be repeated immediately or used excessively.</td>
</tr>
<tr>
<td>4</td>
<td>The first measure should be a half rest followed by a half note.</td>
</tr>
<tr>
<td>5</td>
<td>The penultimate measure is a tied quarter note, followed by two eight notes and a half note.</td>
</tr>
<tr>
<td>6</td>
<td>The last measure should be a whole note.</td>
</tr>
<tr>
<td>7</td>
<td>Ties are allowed between measures and notes of the same pitch.</td>
</tr>
<tr>
<td>8</td>
<td>A half note can be tied to a half note or a quarter note.</td>
</tr>
<tr>
<td></td>
<td>No other ties are possible.</td>
</tr>
<tr>
<td>9</td>
<td>Maximum two measures of the same note value (duration) are allowed.</td>
</tr>
<tr>
<td></td>
<td>Variations with eight notes do not count.</td>
</tr>
</tbody>
</table>

In the next section, these hard rules are used by the variable neighbourhood search algorithm as feasibility criteria. The objective function, based on the soft rules, will be minimized by the algorithm.

2.4 Variable Neighbourhood Search

Most of the existing literature on CAC proposes population-based algorithms (especially genetic/evolutionary algorithms) to compose music. In this chapter, the class of local search metaheuristics is further explored and the variable neighbourhood search algorithm (VNS) previously developed for first species counterpoint is expanded to generate fifth species. The VNS developed in Chapter 1 for CAC significantly outperforms a genetic algorithm.

Given the results of this comparison on first-species counterpoint (a much simpler optimization problem), there is every reason to expect that a similar GA
to automatically compose fifth-species counterpoint will be outperformed by our VNS approach. Moreover, whereas the development of genetic operators (especially crossover) for first-species counterpoint is relatively straightforward, this is not at all the case for fifth-species counterpoint, as the presence of different rhythmic patterns implies that there is no simple note-to-note correspondence between two fragments of fifth-species counterpoint music. Developing a powerful GA to compose fifth-species counterpoint is therefore a complex undertaking that is beyond the scope of this chapter.

The developed VNS starts from an initial random fragment \( s \). Whilst generating this fragment, the hard rules from the previous section are taken into consideration to ensure that the fragment is feasible. This means, among other things, that the rhythmic patterns of the first, penultimate and last measure are set correctly. For each measure, a pattern is chosen from the set of allowed patterns and ties are applied correctly. The pattern selection mechanism ensures that there are no more than two sequential measures with the basic rhythmic pattern. This basic pattern considers the rhythm without ties and eight note decorations. Finally, the pitches of all notes are randomly selected from the correct key. This ensures that the initial fragment \( s \) is feasible and can be used as a starting point for the VNS.

The other components of the algorithm remain largely the same as in the previous chapter and are described in detail in Section 1.3. Figure 1.4 gives an overview of the different components. They include three distinct neighbourhoods with a steepest descent strategy (defined by the same move types as described in the previous chapter), a short term memory structure, a perturbation move and an adaptive weights mechanism. The algorithm stops when either an optimal fragment is found \( f(s) = 0 \) or when the maximum number of successive perturbations without improvement of the best fragment, \( s_{\text{best}} \), have been encountered. This stopping criterion is set by the user through the parameter \( \text{maxiters} \). To set the optimal values for the different parameters of the components, a statistical experiment was conducted in Section 2.6.
2.5. ARCHITECTURE AND IMPLEMENTATION

Table 2.2.: Properties of the Note object

<table>
<thead>
<tr>
<th>Data field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pitch</td>
<td>MIDI value of the pitch of the note.</td>
</tr>
<tr>
<td>duration</td>
<td>Duration expressed in number of beats.</td>
</tr>
<tr>
<td>measure</td>
<td>The number of the measure that the note is in.</td>
</tr>
<tr>
<td>tied</td>
<td>0 if the note is not tied, 1 if it is the start of a tie, 2 if it is the end of a tie.</td>
</tr>
<tr>
<td>beat</td>
<td>Which beat the note falls on (1 to 16).</td>
</tr>
</tbody>
</table>

The VNS algorithm is implemented in C++ as Optimuse (version 2). The previous version of Optimuse optimized first species counterpoint music and only dealt with whole notes. The only information that was needed per note was the MIDI value of its pitch (integer value). Therefore, a musical fragment could be represented as a vector of integers. When dealing with fifth species counterpoint, this representation is no longer valid. A note is now an object with a data field for pitch, duration, measure, tied and beat (see Table 2.2). A musical fragment is a vector which contains all of the note objects in sequence.

The user can specify the input parameters “key” (e.g. G# major) and “weights of the subscores” in a file called input.txt. The parameters of the VNS that are discussed in the next section are set to their optimal value by default and can be overwritten with command line arguments.

To allow the user to easily interact with Optimuse, a plug-in for the open source music notation and playback program MuseScore was written in JavaScript with QtScript Engine. This provides a drop-down menu to access Optimuse from a user-friendly interface. The generated music is displayed on the screen and can be played back in MuseScore. An export function to popular formats such as MIDI, PDF, Lilypond is provided. The music is transferred from Optimuse to MuseScore in the MusicXML format. An XML-based music
The starting point for composing a counterpoint melody is a cantus firmus. The user can either input a new cantus firmus in MuseScore or choose to generate a new one from the Optimuse drop-down menu (see Figure 2.3). The generated cantus firmus is displayed in editable form in MuseScore and can be modified to suit the user’s expectations. When a satisfactory cantus firmus is displayed, the Optimuse drop-down menu can again be used to generate a fitting counterpoint melody.

Optimus version 2.0 (including the MuseScore plug-in) is available for download at http://antor.ua.ac.be/optimus.
The developed VNS algorithm consists of different components, that are described in detail in the Sections 2.4 and 1.3. As is common in metaheuristics, many of these components have one or more parameters that needs to be set, such as the tabu tenure. To thoroughly test the effectiveness of these components and their possible parameter settings, an exhaustive statistical experiment was again performed. Table 2.3 displays the analyzed factors.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
<th>No. of levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_{c1}$ - Swap</td>
<td>on with $tt_{c1} = 0, tt_{c1} = \frac{1}{16}, tt_{c1} = \frac{1}{8}$, off</td>
<td>4</td>
</tr>
<tr>
<td>$N_{c2}$ - Change1</td>
<td>on with $tt_{c2} = 0, tt_{c2} = \frac{1}{16}, tt_{c2} = \frac{1}{8}$, off</td>
<td>4</td>
</tr>
<tr>
<td>$N_{sw}$ - Change2</td>
<td>on with $tt_{sw} = 0, tt_{sw} = \frac{1}{16}, tt_{sw} = \frac{1}{8}$, off</td>
<td>4</td>
</tr>
<tr>
<td>Random move (randsize)</td>
<td>$\frac{1}{4}$ changed, $\frac{1}{8}$ changed, off</td>
<td>3</td>
</tr>
<tr>
<td>Adaptive weights</td>
<td>on, off</td>
<td>2</td>
</tr>
<tr>
<td>Max. number of iterations (maxiters)</td>
<td>5, 20, 50</td>
<td>3</td>
</tr>
<tr>
<td>Length of music (length)</td>
<td>16, 32 measures</td>
<td>2</td>
</tr>
</tbody>
</table>

$t_t = \text{tabu tenure of the tabu list of neighbourhood } N_i$, expressed as a fraction of the total number of notes.

A full factorial experiment was run to test all possible combinations of the factors. This resulted in 2304 runs ($2^2 \times 3^2 \times 4^3$). The cantus firmus that is used as an input for generating the counterpoint is composed by the previous version of Optimuse for each of the 2304 runs. A Multi-Way ANOVA (Analysis of Variance) was estimated with the open source software package R (Bates D., 2012). The model examines the influence of the parameter settings from Table 2.3 on the musical quality of the end fragment as well as the necessary computing time.

An ANOVA model was first calculated to identify the factors with a significant influence on the solution quality. This linear regression model only took into account the main effects. To improve the quality of the model, a second, ANOVA model was constructed taking into account the interaction effects
The $R^2$ statistic of this improved model is 0.98 (see Table 2.4), which means that the model accounts for 98% of the variation around the mean value of the objective function.

<table>
<thead>
<tr>
<th>Measure</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R^2$</td>
<td>0.9836</td>
</tr>
<tr>
<td>$R^2$ Adj</td>
<td>0.9812</td>
</tr>
<tr>
<td>$F$-statistic</td>
<td>410.9 on 293 and 2010 DF</td>
</tr>
<tr>
<td>$p$-value</td>
<td>$&lt; 2.2e^{-16}$</td>
</tr>
</tbody>
</table>

The $p$-values of the factors are displayed in Table 2.5. The interaction effects between more than two factors have been omitted in the table for clarity. The table reveals that all of the factors have a significant influence on the quality of the generated musical fragment ($p < 0.05$), with the exception of the tabu tenure of the change1 neighbourhood. This means that this tabu list does not have a significant influence on the solution quality. Although it is established that the other factors have a significant influence on the result, the nature of this influence still needs to be examined. The mean plots in Figure 2.4 clarify which parameter settings have a positive or negative influence on the result and reveal their optimal settings with respect to the objective function. The interaction plots were also drawn up in R to verify the conclusions from the mean plots for the interaction effects between parameters.

The mean plots for all three neighbourhoods clearly show an improvement of the quality of the best found fragment when the respective neighbourhood is active. The average value for the objective function is significantly lower when the neighbourhoods are activated. This means that all three of the local search neighbourhoods make a positive contribution to the solution quality. The tabu tenure for the change2 and swap neighbourhood also have a significant influence on the quality of the end result. Figure 2.4(e) and 2.4(f) show that a tabu tenure of $\frac{1}{16}$ of the length of the music is optimal. The plot for the size of the permutation (see Figure 2.4(g)) offers another important insight. The
Table 2.5.: Multi-Way ANOVA model with interactions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Df</th>
<th>F value</th>
<th>Prob (&gt;F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_{c1}$</td>
<td>1</td>
<td>9886.2323</td>
<td>&lt; 2.2e$^{-16}$</td>
</tr>
<tr>
<td>$N_{c2}$</td>
<td>1</td>
<td>15690.7234</td>
<td>&lt; 2.2e$^{-16}$</td>
</tr>
<tr>
<td>$N_{sw}$</td>
<td>1</td>
<td>3900.2959</td>
<td>&lt; 2.2e$^{-16}$</td>
</tr>
<tr>
<td>randsize</td>
<td>2</td>
<td>1110.1724</td>
<td>&lt; 2.2e$^{-16}$</td>
</tr>
<tr>
<td>maxiters</td>
<td>2</td>
<td>322.6488</td>
<td>&lt; 2.2e$^{-16}$</td>
</tr>
<tr>
<td>length</td>
<td>1</td>
<td>165.6053</td>
<td>&lt; 2.2e$^{-16}$</td>
</tr>
<tr>
<td>adj. weights</td>
<td>1</td>
<td>4.0298</td>
<td>0.0448367</td>
</tr>
<tr>
<td>$tt_{c1}$</td>
<td>2</td>
<td>2.2575</td>
<td>0.1048971</td>
</tr>
<tr>
<td>$tt_{c2}$</td>
<td>2</td>
<td>8.271</td>
<td>0.0002646</td>
</tr>
<tr>
<td>$tt_{sw}$</td>
<td>2</td>
<td>3.2447</td>
<td>0.0391833</td>
</tr>
<tr>
<td>$N_{c1}$:$N_{c2}$</td>
<td>1</td>
<td>25198.1739</td>
<td>&lt; 2.2e$^{-16}$</td>
</tr>
<tr>
<td>$N_{c1}$:$N_{sw}$</td>
<td>1</td>
<td>10264.8395</td>
<td>&lt; 2.2e$^{-16}$</td>
</tr>
<tr>
<td>$N_{c2}$:$N_{sw}$</td>
<td>1</td>
<td>10753.1655</td>
<td>&lt; 2.2e$^{-16}$</td>
</tr>
<tr>
<td>$N_{c1}$:randsize</td>
<td>2</td>
<td>114.2753</td>
<td>&lt; 2.2e$^{-16}$</td>
</tr>
<tr>
<td>$N_{c2}$:randsize</td>
<td>2</td>
<td>415.5001</td>
<td>&lt; 2.2e$^{-16}$</td>
</tr>
<tr>
<td>$N_{sw}$:randsize</td>
<td>2</td>
<td>117.0946</td>
<td>&lt; 2.2e$^{-16}$</td>
</tr>
<tr>
<td>$N_{c1}$:maxiters</td>
<td>2</td>
<td>19.0193</td>
<td>6.564e$^{-09}$</td>
</tr>
<tr>
<td>$N_{c2}$:maxiters</td>
<td>2</td>
<td>67.8841</td>
<td>&lt; 2.2e$^{-16}$</td>
</tr>
<tr>
<td>$N_{sw}$:maxiters</td>
<td>2</td>
<td>19.1179</td>
<td>5.959e$^{09}$</td>
</tr>
<tr>
<td>randsize:maxiters</td>
<td>4</td>
<td>42.7951</td>
<td>&lt; 2.2e$^{-16}$</td>
</tr>
<tr>
<td>$N_{c1}$:length</td>
<td>1</td>
<td>0.7116</td>
<td>0.3989975</td>
</tr>
<tr>
<td>$N_{c2}$:length</td>
<td>1</td>
<td>0.7192</td>
<td>0.3965164</td>
</tr>
<tr>
<td>$N_{sw}$:length</td>
<td>1</td>
<td>7.0076</td>
<td>0.0081798</td>
</tr>
<tr>
<td>randsize:length</td>
<td>2</td>
<td>1.5681</td>
<td>0.2086959</td>
</tr>
<tr>
<td>maxiters:length</td>
<td>2</td>
<td>4.0212</td>
<td>0.0180768</td>
</tr>
<tr>
<td>$N_{c1}$:adj. weights</td>
<td>1</td>
<td>96.6782</td>
<td>&lt; 2.2e$^{-16}$</td>
</tr>
<tr>
<td>$N_{c2}$:adj. weights</td>
<td>1</td>
<td>45.0273</td>
<td>2.519e$^{11}$</td>
</tr>
<tr>
<td>$N_{sw}$:adj. weights</td>
<td>1</td>
<td>0.0005</td>
<td>0.9813378</td>
</tr>
<tr>
<td>randsize:adj. weights</td>
<td>2</td>
<td>233.8511</td>
<td>&lt; 2.2e$^{-16}$</td>
</tr>
<tr>
<td>maxiters:adj. weights</td>
<td>2</td>
<td>0.8878</td>
<td>0.4117166</td>
</tr>
<tr>
<td>length:adj. weights</td>
<td>1</td>
<td>13.0046</td>
<td>0.0003183</td>
</tr>
</tbody>
</table>
random jump has the best effect on the solution quality when 12.5% of the notes are changed. The interaction plots support these conclusions.

Another significant means plot is that of the adaptive weights mechanism (see Figure 2.4(h)). The mean value of the objective function is slightly better when this mechanism is functional. A more detailed study of interaction plots reveals that the adaptive weights mechanism makes a bigger positive contribution for fragments of a smaller length. Finally, the last plot (see Figure 2.4(i)) shows that a higher number of allowed maximum iterations produces a better result.

A similar ANOVA model has been constructed to analyze the computing time of the algorithm. Again all factors have a significant influence, except for the tabu tenure of the change1 neighbourhood and the adaptive weights mechanism. The means plots in Figure 2.4 reveal the nature of their influence. The computing time mostly has an inverse relationship with the solution quality, which is due to the nature of the stopping criteria. Whenever the search gets stuck in a local optimum, the quality will remain poor, but the algorithm’s stopping criteria will be met sooner. This way good components of the algorithm (e.g. the random jump) cause an increase in computing time, because the search for better solutions can continue for a longer time, often resulting in a better solution quality.

An overview of the optimal parameter settings is given in Table 2.6. Improvements in solution quality was the first criterion in determining these optimal settings.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>(N_{sw}) - Swap</td>
<td>on with (tt_{sw} = \frac{1}{16})</td>
</tr>
<tr>
<td>(N_{c1}) - Change1</td>
<td>on with (tt_{c1} = \frac{1}{16})</td>
</tr>
<tr>
<td>(N_{c2}) - Change2</td>
<td>on with (tt_{c2} = \frac{1}{16})</td>
</tr>
<tr>
<td>Random move</td>
<td>(\frac{1}{5}) changed</td>
</tr>
<tr>
<td>Adaptive weights</td>
<td>on</td>
</tr>
<tr>
<td>Max. number of iterations</td>
<td>50</td>
</tr>
</tbody>
</table>
2.6. Experiments

(a) Swap neighbourhood

(b) Change1 neighbourhood

(c) Change2 neighbourhood

(d) Tabu tenure of Swap

(e) Tabu tenure of Change1

(f) Tabu tenure of Change2

(g) Size of perturbation

(h) Adaptive weights procedure

(i) Maximum iterations

Figure 2.4: Mean plots CP
The VNS algorithm with the optimal settings was run on a fragment consisting of 32 measures. The evolution of the value of the objective function, both with original and adapted weights, is displayed in Figure 2.5. This plot shows a steep improvement of the solution quality during the first 100 moves, followed by a more gradual improvement in the next 600 moves. The many fluctuations of the score are due to the perturbation moves. Whenever a local optimum is reached, a temporary increase in the objective score leads to an eventual decrease. This confirms the importance of the perturbation move.

![Figure 2.5: Evolution over time with optimal parameter settings](image)

Figure 2.6 shows an example of a fifth species counterpoint fragment generated by Optimuse. The objective score of this counterpoint fragment is 0.556776. In comparison, random initial fragments typically have a score of around 10. Pdf scores and mp3 examples of Optimuse’s output are available on [http://antor.ua.ac.be/optimuse](http://antor.ua.ac.be/optimuse).

It is the subjective opinion of the authors that the generated fragment sounds pleasing to the ear. Yet it can not be considered to be a finished composition. One of the reasons for that is its lack of theme or sense of direction. This is addressed later on in Chapter 6. The generated music could however, even at this point, be used by a composer as a starting point of a composition.
A VNS algorithm was developed and optimized that can generate fifth species counterpoint music based on a cantus firmus. The rules of fifth species counterpoint were quantified and used as an objective function for this algorithm. The different components of the VNS were thoroughly tested and analyzed by means of a full factorial experiment. This revealed the significant components and their optimal parameter settings. The resulting algorithm was implemented as a C++ software called Optimuse, including a plug-in for MuseScore which provides a user-friendly environment for interacting with the VNS. The musical fragments composed by Optimuse can reach good values for the objective function (see Figure 2.5) and sound pleasing, at least to the subjective ear of the author. In it’s current state, the system might offer composers an original starting point for their compositions.

The next chapter will discuss how the VNS from this chapter was implemented as an Android application and modified to generate a continuous stream of music.
3

FUX, AN ANDROID APP THAT GENERATES COUNTERPOINT

This chapter is based on the paper “D. Herremans, K. Sørensen. 2013. FuX, an Android app that generates counterpoint. IEEE Symposium on Computational Intelligence for Creativity and Affective Computing (CICAC). 48-55.”
A variable neighbourhood search (VNS) algorithm was developed and implemented as a C++ software called Optimuse in the previous chapters. This algorithm can efficiently generate musical fragments of a pre-specified length on a pc. In this research the existing VNS algorithm is modified to generate a continuous stream of new music. It is then ported to the Android platform. The resulting Android app, called FuX, is user friendly and can be installed on any Android phone or tablet. Possible uses include playing an endless stream of classical music to babies. Babies often calm down and experience health benefits from listening to soothing music (Schwartz and Ritchie, 2004). It can be conjectured that the highly consonant style of classical counterpoint is especially suitable for this purpose. Moreover, parents who are tired of listening to the same tune thousands of time might prefer the non-repetitiveness of the music generated by FuX. FuX also provides an endless stream of royalty-free music that could be played in elevators, lobbies and as call centre waiting music. Finally, it might offer an endless source of inspiration to composers.

3.1 INTRODUCTION

In order to make the implementation of the VNS accessible and easily usable for a large audience, an Android application (or app) called FuX is developed in this research. Android is a software toolkit that runs on a large number of mobile devices. Mobile phones and tablets have never been more popular and are getting increasingly more powerful (Meier, 2012).

There are a plethora of other mobile operating systems available. Symbian from Nokia, Windows Mobile from Microsoft, BlackBerry from RIM, iOS from Apple etc. According to a Survey of Oliver (2009) none of these operating systems (including Android) are perfect for developers. The two most used operating systems are iOS and Android (Goadrich and Rogers, 2011). The VNS developed in this research is implemented on the Android system, which allows it to run on a multitude of devices, not only those from Apple, with many of these devices available at a relatively low cost. An added advantage is Android’s “open” nature and large support community compared to iOS’s
lack of developer tools (Oliver, 2009). Google reported that more than 500 million Android devices have been activated (Barra, September 2012).

When exploring Google Play\(^1\), the web based platform to easily install new Android applications, the category “music” displays thousands of entries. Many applications have been developed to play music (Yong-Cai et al., 2010), recommend music based on a user profile (Kaminskas and Ricci, 2012) or a travel location (Braunhofer et al., 2011), assist in browsing large music libraries (Tzanetakis et al., 2009), finding music by singing/humming (Park and Chung, 2012), and many more. Park and Chung (2012) give an extensive overview of music related Android applications.

Since Android 1.0 was only released in 2008 (Google, 2012a), the number of publications on the use of metaheuristics implemented on this platform is still limited. Added to that, the trend to invent different names for similar existing metaheuristics makes it harder to get an overview of the entire field (Sörensen, 2013). Fajardo and Oppus (2010) have implemented a genetic algorithm for mobile disaster management. Zheng et al. (2012) use simulated annealing for WiFi based indoor localization on Android.

In the next section, the VNS and how it was modified to enable continuous generation is discussed. Section 3.3 explains the implementation (called FuX) of the VNS for the Android platform.

3.2 FROM OPTIMUSE TO FUX

The VNS used in this research operates in two phases. In the first phase, the cantus firmus is generated. After that, the counterpoint is composed on top of this cantus firmus. The algorithm used to generate the melodies in both phases is identical, it only differs in the objective function that is used. The cantus firmus is evaluated by the objective function that focuses only on melodic rules. For the (fifth species) counterpoint melody both melodic and harmonic rules are evaluated. This two-phased design originated from the fact that a

\(^1\) http://play.google.com
counterpoint melody is usually composed “against” an existing cantus firmus
and also allows a user to input her own cantus firmus, at least in the original
Optimuse implementation described in the previous chapters (Herremans
and Sörensen, 2013). An overview of the size of the neighbourhood is given
in Table 3.1 and is dependent on the length of the fragment $L$ that being
generated.

<table>
<thead>
<tr>
<th>$N_i$</th>
<th>Name</th>
<th>Description</th>
<th>Neighbourhood size</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_1$</td>
<td>Change1</td>
<td>Change one note</td>
<td>$16 \times 9 \times L$</td>
</tr>
<tr>
<td>$N_2$</td>
<td>Change2</td>
<td>Change two sequential notes</td>
<td>$16 \times 9 \times 9 \times L$</td>
</tr>
<tr>
<td>$N_3$</td>
<td>Swap</td>
<td>Swap two notes</td>
<td>$(16 \times L)$</td>
</tr>
</tbody>
</table>

$L$ is the length of the fragment expressed in units of 16 notes.

Figure 3.1 visualizes the developed VNS implemented in the Android applica-
tion. Only the stopping criterium is different from the one used in Optimuse
as there is no maximum number of iterations. The VNS will keep improving
the solution until the maximum time limit is reached or the optimal solution
$f(s) = 0$ is reached.

The original VNS implementation is able to compose fragments of any length,
as long as they are a multiple of 16 measures. Now the VNS needs to be able
to sequentially generate new fragments that can be considered as one large
fragment. The implementation was therefore slightly modified. The VNS is
now able to generate 16 measures with a time limit $t_1$. This generated fragment
is used as the starting fragment of the continuously generated piece.

The subsequent fragments that are generated consist of 8 measures and are
generated with a time limit $t_2$. However they are evaluated by also taking into
account the last 8 measures of the previous fragment. This means that the
VNS always evaluates the last 16 measures. Doing so ensures that no “breaks”
in the music occur.

FuX needs to be able to sequentially generate fragments that are played live.
Thus, the speed of the algorithm becomes increasingly important, especially
since mobile devices often have limited resources such as low-power CPUs,
3.3 Android Implementation

Android is a software toolkit for mobile phones based on the Linux platform developed by Google and the Open Handset Alliance. At the bottom of the Android software stack is the Linux operating system (Kernel 2.6), this provides all basic system functionality such as memory management and limited RAM and slow I/O (Oliver, 2009). In order to speed up the VNS, the order of the neighbourhoods was changed from the previous implementation to the one listed in Table 3.1. The new order described in this chapter favours the smaller neighbourhoods because they are often able to make large improvements in the beginning of the run, which ensures that a “reasonable” quality can be obtained fairly quickly.
Chapter 3. Fux, an Android App that Generates Counterpoint

device drivers (Mongia and Madisetti). On top of the OS, there is a set of native libraries written in C/C++ that offer, for instance, audio and video support (Google, 2012a). The next step in the Android stack contains the runtime engine – the Dalvik Virtual Machine (VM). Dalvik runs applications written in Android’s variant of java (Bornstein, 2008).

Android developers can use the Android Software Development Kit (SDK), to get access to the same framework that is used by the core applications. These powerful libraries allow the development of a wide range of java based applications (Google, 2012a).

Since resources are typically limited on mobile devices, a careful consideration had to be made on how to implement the VNS. Son and Lee (2011) recommend the use of Android Native Development Kit (NDK) for computationally expensive tasks. This is confirmed by benchmark experiments (Lin et al., 2011). Android NDK provides a native development platform that allows embedding components that use native code. With NDK, developers can compile C/C++ code for the Android development platform (Ratabouil, 2011). Since the previously developed code for the VNS algorithm was in C++, this code could be slightly altered and integrated in the Android app. More details on the original C++ code are described in the previous chapters.

3.3.1 Continuous generation

The app developed in this research can continuously generate counterpoint using a VNS. This is achieved by iteratively generating small MIDI files and playing them consecutively. Since the music is played by the device as it is being generated, there should be (at least) two threads running at the same time. A “generate” thread (thread1) and a “playback” thread (thread2). This multithreading approach is described in Table 3.2. When the app is initialized the VNS algorithm generates the first 16 measures. These are saved as a MIDI file. Whenever the user presses “Play”, thread1 generates the next 8 measures whilst thread2 plays the first MIDI file. Directly after the first MIDI file finishes playing, the second MIDI file is played. If the file is not ready
Table 3.2: Multithreading

<table>
<thead>
<tr>
<th>Time</th>
<th>Generate</th>
<th>Playback</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>16 measures (file 1)</td>
<td></td>
</tr>
<tr>
<td>8s</td>
<td>8 measures (file 2)</td>
<td>file 1</td>
</tr>
<tr>
<td>16s</td>
<td>8 measures (file 3)</td>
<td>file 2</td>
</tr>
<tr>
<td>24s</td>
<td>8 measures (file 4)</td>
<td>file 3</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

yet, thread2 waits for thread1 to finish the generation process. This should be avoided, since it causes an interruption in the playback. This process is repeated until the user pauses or stops it.

The time cutoff for the VNS algorithm is currently set to 10 seconds for the initial generation. This time is divided between the generation of the cantus firmus (3 seconds) and the counterpoint (7 seconds). Because of the complexity of the counterpoint, more time was allotted to its generation. When the cantus firmus reaches an optimum before 3 seconds are passed, the remaining CF time is added to the CP time. The total time is divided by taking into account the following relationship $t_{cp} = 2 \times t_{cf} + 1$. This formula doubles the generation time for the counterpoint and adds one second to fully exploit the available time. Another relationship might also work, as long as $t_{cp}$ is significantly larger than $t_{cf}$.

For the generation of the 8 measure fragments, 7 seconds are allotted. This time is divided with the same formula: 2 seconds for the CF and 5 seconds for the CP. The speed of the MIDI playback is set to 1 beat per second. This means that the total time available for generating the file is 8 seconds. The VNS algorithm uses 7 seconds to generate the fragment, which means that 1 second is available as a buffer for actually writing the MIDI file.
3.3.2 MIDI files

The VNS algorithm is executed in C++ and returns a native java array. The newly generated music is contained entirely in the `jarray`. This `jarray` is converted to a MIDI file using the library `Android-Midi-Lib` (Leffelman, 2012). The MIDI files are stored in the cache folder of the device, so that they are automatically removed periodically. When the VNS is run to generate the next fragment, the previous `jarray` is passed as input to the VNS algorithm, so that it can take into account the previous 8 measures when evaluating the next musical fragment.

Android’s `MediaPlayer` class is used to play the MIDI files. The `OnCompletionListener` of this class offers a way to easily play the next MIDI file when playback is finished. Although a small delay between the files might be heard on older Android devices, version 4.1 (Jelly Bean) advertises audio chaining as one of its features (Google, 2012b). This low latency audio playback enables the files to be played continuously as if they were one big file.

3.3.3 Implementation and results

Figures 3.2(a) and 3.2(b) show the evolution over time of the objective function for cantus firmus and counterpoint. These results were obtained by using an Eclipse Android Virtual Device with Android 4.0.3, ARM processor and 512MB RAM. The emulator was installed on an OpenSuse system with Intel®Core™2 Duo CPU@ 2.20GHz and 3.8GB RAM. The results show a fairly steady improvement of the objective function that lessens somewhat over time. When generating the second file, the initial objective score is better than when generating file 1. This can be explained by the fact that the initial fragment is based on 16 randomly generated measures. The second file is formed by 8 “optimized” measures followed by 8 randomly generated measures, which causes the starting score to be better. This is confirmed by Figures 3.2(a) and 3.2(b). When generating file 2 a better end score can be found than with file 1 despite the lower cutoff times. The maximum cutoff
3.3. ANDROID IMPLEMENTATION

Figure 3.2.: Evolution of the objective function over time
time of the algorithm is respectively 2 and 3 seconds for CF and 7 and 10 seconds for CP, as described in section 3.3.1. Figure 3.2(a) shows that the algorithm is able to find an optimal cantus firmus before the maximum time is reached. This allows the generation of the counterpoint to begin sooner (after 1.5 and 2 seconds respectively), thus expanding the time that the VNS can use to generate CP.

The quality of the generated music depends highly on the architecture of the mobile device on which it is installed. The results described in the previous paragraph confirm that the optimal objective score is reached for the cantus firmus. There is also a significant improvement of the objective score for counterpoint. While the objective score only measures how well the generated music fits into the counterpoint style, it is the subjective opinion of the authors that music sounds pleasant to the ear even on lower-end devices. The reader is invited to install the app and listen to the results of this research.

While the music generated by FuX can be considered to largely adhere to the counterpoint rules, it would be interesting to expand the objective function in future versions. Human baroque composers often base their work on the principles of counterpoint, but a finished composition has an encompassing theme and mixes the counterpoint rules with a composer’s creative freedom. It could be argued that the fact that FuX does not find the optimal solution, can be interpreted as a random creative input. Still, an interesting future improvement could be to add more complex rules to the objective function, thus endorsing for instance a recurring theme and more structure, making the generated music sound more like a complete and coherent composition.

An official .apk application package has been generated called FuX. This package is freely available through Google Play at http://play.google.com and can be installed on any Android phone from version 2.1 and up. The user interface for FuX version 1.0 is simple but functional (see Figure 3.3). This user interface is expanded in Chapter 4 to allow a user to specify more options such as playback instrument and composing music with characteristics of a certain composer.
A user-friendly Android application was implemented that can *continuously* play a stream of new counterpoint music. The implemented app, FuX, uses a variable neighbourhood search algorithm to generate the music. The VNS is based on a similar algorithm that generates musical fragments of a pre-specified length on a pc. The original algorithm was adapted to allow the *continuous* generation of music. In order to evaluate the quality of a fragment, a quantification of the extensive rules of Fux was used. This resulted in an Android app with a user-friendly interface that can generate a continuous stream of music that sounds pleasing to the ear.
The next part will focus on generating music without having a predefined objective function. In Chapter 4 a large existing database of music will be analysed, in order to find criteria specific to a certain style or composer. These are then implemented in the objective function. This will allow the generation of music with composer-specific characteristics without having to rely on the existence of predefined rules from music theory. This is combined with improvements to FuX’s user interface so that the user can choose the instrument and modify the objective function using sliders. In Chapter 5 the learning process is expanded to modelling counterpoint and how we can efficiently generate music that is preferred according to these learned statistical models. In Chapter 6, different quality assessment metrics based on these statistical models are evaluated and music is generated within a certain structure, to ensure the generation of more complete compositions.
Part 2

MUSIC GENERATION WITH MACHINE LEARNING
LOOKING INTO THE MINDS OF BACH, HAYDN AND BEETHOVEN: CLASSIFICATION AND GENERATION OF COMPOSER-SPECIFIC MUSIC

This chapter is based on the paper “D. Herremans, K. Sørensen, D. Martens. 2014. Looking into the minds of Bach, Haydn and Beethoven: Classification and generation of composer-specific music. Working paper 2014001. Faculty of Applied Economics, University of Antwerp.”
The task of recognizing a composer by listening to a musical fragment used to be reserved for experts in music theory. The question that is tackled in this research is “Can a computer accurately recognize who composed a musical piece?”. We take a data-driven approach, by scanning a large database of existing music and develop three classification models that can accurately classify a musical piece in groups of three composers. This research builds predictive classification models that can be used both for theory-building and to calculate the probability that a piece is composed by a certain composer.

The first goal of this chapter is to build a ruleset and a decision tree that gives the reader an understanding of the differences between styles of composers (Bach, Haydn and Beethoven). These models give the reader more insight into why a piece belongs to a certain composer. The second goal is to build more accurate classification models that can help an existing music composition algorithm generate composer-specific music, i.e., music that contains characteristics of a specific composer. In Part 1, a variable neighbourhood search algorithm (VNS) was developed that can compose counterpoint music. The logistic regression model developed in this chapter is incorporated into the objective function of this VNS. The resulting system is able to play a stream of continuously generated contrapuntal music with composer-specific traits.

4.1 Prior work

The digitization of the music industry has attracted growing attention to the field of Music Information Retrieval (MIR). MIR is a multidisciplinary domain, concerned with retrieving and analysing multifaceted information from large music databases (Downie, 2003). According to Byrd and Crawford (2002) the first publication about MIR originates from the mid-1960s (Kassler, 1966). Kassler (1966) uses the term MIR to name the programming language he developed to extract information from music files. In recent years, numerous MIR systems have been developed and applied to a broad range of topics. An interesting example is the content-based music search engine “Query by Humming” (Ghias et al., 1995). This MIR system allows the user to find a song based on a tune that he or she hums. Another application of MIR is measuring
the similarity between two musical pieces (Berenzweig et al., 2004). In this research however, the focus lies on using MIR for composer classification.

When it comes to automatic music classification, machine learning tools are used to classify musical pieces per genre (Tzanetakis and Cook, 2002; Conklin, 2013b), cultural origin (Whitman and Smaragdis, 2002), mood (Laurier et al., 2008) etc. While the general task of automatically classifying music per genre has recently received increasing attention, see Conklin (2013b) for a more complete overview, the more specific task of composer classification remains largely unexplored (Geertzen and van Zaanen, 2008).

A system to classify string quartet pieces by composer has been implemented by Kaliakatsos-Papakostas et al. (2011). In this system, the four voices of the quartets are treated as a monophonic melody, so that it can be represented through a discrete Markov chain. The weighted Markov chain model reaches a classification success of 59 to 88% in classifying between two composers. The Hidden Markov Models designed by Pollastri and Simoncelli (2001) for the classification of 605 monophonic themes by five composers has a lower accuracy rate. Their best result has an accuracy of 42% of successful classifications on average. However, it must be noted that this accuracy is not measured for classification between two classes like in the previous example, but classification is done with five classes or composers.

Wołkowicz et al. (2007) show that another machine learning technique, i.e., n-grams, can be used to classify piano files in groups of five composers. An n-gram model tries to find patterns in properties of the training data. These patterns are called n-grams, in which n is the number of symbols in a pattern. Hillewaere et al. (2010) also use n-grams and global feature models to classify string quartets for two composers (Haydn and Mozart). Their trigram approach to composer recognition of string quartets has a classification accuracy of 61.4%, for violin and viola, and 75.4% for cello.

n-grams belong to the family of grammars, a group of techniques that use a rule-based approach to specify patterns (Searls et al., 2002). Buzzanca (2002) states that the use of grammars such as n-grams for modelling music style is unsatisfying because they are vulnerable with regard to creating ad hoc rules and they can not represent ambiguity in the musical process. Buzzanca
(2002) works with Palestrina style recognition, which could be considered a more general problem than composer recognition. Instead of n-grams, he implemented a neural network that can classify with 97% accuracy on the test set. Although it should be noted that all the pieces of the music database are heavily preprocessed and classification is only done on short “main themes”. One of the disadvantages of neural networks is that these models are in essence a black-box, as they provide a complex non-linear output score. They do not give any new music theoretical insights in the differences between two composers as they are. To generate a comprehensible model, rules could be extracted from an existing black-box neural network, using pedagogical rule extraction techniques like Trepan and G-REX (Martens et al., 2007).

Van Kranenburg and Backer (2004) apply other types of machine learning algorithms to a database of 320 pieces from the eighteenth and early nineteenth century. 20 high-level style markers based on properties of counterpoint are examined. The K-means clustering algorithm they developed shows that musical pieces of the chosen five composers do form a cluster in feature space. A decision tree (C4.5) and nearest neighbour classification algorithm show that it is possible to classify pieces with a fairly low error rate. Although the features are described in the paper, a detailed description of the models is missing.

Mearns et al. (2010) also use high-level features based on counterpoint and intervallic features to classify similar musical pieces. Their developed C4.5 decision tree and naive Bayes models correctly classified 44 out of 66 pieces with 7 groups of composers. Although the actual decision tree is not displayed in the paper it could give music theorists an insight in the differences between styles of composers.

In the next sections, a technique is described to extract useful musical features from a database. These features are then used to build four accurate classification models. In contrast to many existing studies, the models described in this research are both accurate as well as comprehensible and the full details are described in this chapter. The developed models give insights into the styles of Haydn, Beethoven and Bach. In a next phase, one of the models is incorporated in the VNS algorithm developed in Part 1, which results in a
system that is able to generate music that has characteristics of a specified composer.

4.2 Feature Extraction

Traditionally, a distinction between symbolic MIR and audio MIR is made. Symbolic music representations, such as MIDI, contain very high-level structured information about music, e.g., which note is played by which instrument. However, most existing work revolves around audio MIR, in which automatic computer audition techniques are used to extract relevant information from audio signals (Tzanetakis et al., 2003). Different features can be examined depending on the type of audio file that is being analysed. These features can be roughly classified in three groups:

- **low-level features** extracted by automatic computer audition techniques from audio signals such as WAV files, e.g., spectral flux and zero-crossing rate (Tzanetakis et al., 2003)

- **high-level features** extracted from structured files such as MIDI, e.g., interval frequencies, instrument presence and number of voices (McKay and Fujinaga, 2006).

- **metadata** such as factual and cultural information information related to a file which can be both structured or unstructured, e.g., play list co-occurrence (Casey et al., 2008)

It is not an simple task to extract note information from audio recordings of polyphonic music (Gómez Gutiérrez, 2006). Since the high-level features used in this research require detailed note information, we chose to work with MIDI files. Symbolic files such as MIDI files are in essence the same as musical scores. They describe the start, duration, volume and instrument of each note in a musical fragment and therefore allow the extraction of characteristics that might provide meaningful insights to music theorists. It must be noted that MIDI files do not capture the full richness of a musical performance like audio
files do (Lippincott, 2002). They are, however, very suitable for the features analysed in this research.

4.2.1 KernScores database

The KernScores database is a large collection of virtual music scores made available by the Center for Computer Assisted Research in the Humanities at Stanford University (CCARH). It holds a total of 7,866,496 notes and is available online (CCARH, 2012). This database was specifically created for computational analysis of musical scores (Sapp, 2005). The composers Johann Sebastian Bach, Ludwig van Beethoven and Franz Joseph Haydn were selected for inclusion in our classification models because a large number of musical pieces is available for these three composers in the KernScores database. Having a large amount of instances available per composer allows the creation of more accurate models. 1045 musical pieces from a total of three composers were downloaded from the database. Almost all available musical pieces per composer were selected, except for a few very short fragments. An overview of the selected pieces is given in Table 4.1.

<table>
<thead>
<tr>
<th>Composer</th>
<th># Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Haydn (HA)</td>
<td>254</td>
</tr>
<tr>
<td>Beethoven (BE)</td>
<td>196</td>
</tr>
<tr>
<td>Bach (BA)</td>
<td>595</td>
</tr>
</tbody>
</table>

The KernScores database contains musical pieces in the **KERN notation, ABC notation and MIDI. For this research, the MIDI files are used as they are compatible with the feature extraction software jSymbolic. jSymbolic is a part of jMIR, a toolbox designed for automatic music classification (Mckay and Fujinaga, 2009). Van Kranenburg and Backer (2004) point out that MIDI files are the representation of a performance and are therefore not always an accurate representation of the score. It is true that MIDI files are often recorded by a human playing the score, or can be derived from audio files.
which results in inaccurate timing. However, since the KernScore database is encoded by hand from **KERN files, it offers a reliable source of accurate MIDI files.

### 4.2.2 Implementation of feature extraction

The software used to extract the features is jSymbolic. jSymbolic is a Java based Open Source software that allows easy extraction of high-level features from MIDI files (McKay and Fujinaga, 2007). Twelve features (see Table 4.2) are extracted from our dataset. All of these features offer information regarding melodic intervals and pitches. They are measured as occurrence frequencies normalized to range from 0 to 1.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Feature description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>Chromatic Motion Frequency - Fraction of melodic intervals corresponding to a semi-tone.</td>
</tr>
<tr>
<td>$x_2$</td>
<td>Melodic Fifth Frequency</td>
</tr>
<tr>
<td>$x_3$</td>
<td>Melodic Octaves Frequency</td>
</tr>
<tr>
<td>$x_4$</td>
<td>Melodic Thirds Frequency</td>
</tr>
<tr>
<td>$x_5$</td>
<td>Most Common Melodic Interval Prevalence</td>
</tr>
<tr>
<td>$x_6$</td>
<td>Most Common Pitch Prevalence</td>
</tr>
<tr>
<td>$x_7$</td>
<td>Most Common Pitch Class Prevalence</td>
</tr>
<tr>
<td>$x_8$</td>
<td>Relative Strength of Most Common Intervals - fraction of intervals belonging to the second most common / most common melodic intervals</td>
</tr>
<tr>
<td>$x_9$</td>
<td>Relative Strength of Top Pitch Classes</td>
</tr>
<tr>
<td>$x_{10}$</td>
<td>Relative Strength of Top Pitches</td>
</tr>
<tr>
<td>$x_{11}$</td>
<td>Repeated Notes - fraction of notes that are repeated melodically</td>
</tr>
<tr>
<td>$x_{12}$</td>
<td>Stepwise Motion Frequency</td>
</tr>
</tbody>
</table>

Pitch refers to an absolute pitch, e.g., C in the 7th octave. Pitch class refers to a note without the octave, e.g., C.
The examined featureset is deliberately kept small to avoid overfitting the model (Gheyas and Smith, 2010). McKay and Fujinaga (2006) refer to the “curse of dimensionality”, whereby the number of labelled training and testing samples needed increases exponentially with the number of features. Not having too many features allows a thorough testing of the model with limited instances and can thus improve the quality of the classification model (McKay and Fujinaga, 2006). In this research, a selection of features was made from the 111 features available in jSymbolic. During this selection process, one dimensional features that output frequency information related to intervals or pitches were preferred because of their normalized nature and ease to handle. All features dependent upon the key of the piece or nominal features were omitted. Features related to instruments, such as ‘Electric guitar fraction’, were omitted since they are not relevant for the chosen corpus. The resulting 12 features are displayed in Table 4.2.

jSymbolic outputs the extracted features of all instances in ACE XML files. These XML files are converted to the Weka ARFF format with jMIRUtilities, another tool from the jMIR toolbox (Mckay and Fujinaga, 2009). In the next sections, four classification models are developed based on the extracted data.

### 4.3 Composer Classification Models

Shmueli and Koppius (2011) point out that predictive models can not only be used as practically useful classification models, but can also play a role in theory-building and testing. In this chapter, models are built for two different purposes, based on the aforementioned.

The first objective of this research is to develop a model from which insight can be gained into the characteristics of musical pieces composed by a certain composer. This resulted in a ruleset built with the “Repeated Incremental Pruning to Produce Error Reduction algorithm” (RIPPER) and a C4.5 decision tree. The second objective is to build predictive models (logistic regression and support vector machines) that can accurately determine the probability
that a musical piece belongs to a certain composer. One of these models is then incorporated into the existing objective function of the music generation algorithm, leading to a new metric that allows it to automatically assess how well a generated musical piece fits into a certain composer’s style. In order to accurately modify the developed model for inclusion in the objective function, not all classification models are suited. A logistic regression model was chosen for this purpose. Its implementation is described in the Section 4.4.

A number of machine learning methods like neural networks, Markov chains and clustering have already been implemented for musical style modelling (Dubnov et al., 2003b). Since most of the research papers do not give an accurate description of the model, nor a full feature list, the existing research could only be used as inspiration for the developed models.

Based on the features extracted in the previous section, four supervised learning algorithms are applied to the dataset. Since our dataset includes labelled instances, supervised learning techniques can be used to learn a classification model based on these labelled training instances. The Open Source software Weka is used to create the classification models (Witten and Frank, 2005). Weka offers a toolbox and framework for machine learning and data mining that is recognized as a landmark system in this field (Hall et al., 2009).

In this section, four classifier models are developed with RIPPER, C4.5, logistic regression and support vector machines. The first two models are of a more linguistic nature and therefore more comprehensible (Martens et al., 2011). The other two models are not as comprehensible, but have a better performance. One of these latter models is integrated in the objective function of the music generation algorithm in the next section. The performance results based on accuracy and area under the receiver operating curve (AUC) of all four models are displayed in Table 4.3. Although the distribution is not heavily skewed (see Table 4.1), it is not completely balanced either. Because of this the use of the accuracy measure to evaluate our results is not suited and the AUC was used instead (Fawcett, 2004), yet both are displayed in Table 4.3 to be complete.
The tests were run 10 times, each time with stratified 10-fold cross validation (10CV). During the cross validation procedure, the dataset is divided into 10 folds. 9 of them are used for model building and 1 for testing. This procedure is repeated 10 times. The displayed AUC and accuracy are the average results over the 10 test sets and the 10 runs. The resulting model is built on the entire dataset and can be expected to have a performance which is at least as good as the 10CV performance. A Wilcoxon signed-rank test is conducted to compare the performance of the models with the best performing model. The null hypothesis of this test states: “There is no difference in the performance of a model with the best model”.

Table 4.3.: Evaluation of the models with 10-fold cross-validation

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIPPER ruleset</td>
<td>77%</td>
<td>78%</td>
</tr>
<tr>
<td>C4.5 Decision tree</td>
<td>79%</td>
<td>83%</td>
</tr>
<tr>
<td>Logistic regression</td>
<td>83%</td>
<td>92%</td>
</tr>
<tr>
<td>Support vector machines</td>
<td><strong>86%</strong></td>
<td><strong>93%</strong></td>
</tr>
</tbody>
</table>

\[ p < 0.01: \text{italic}, p > 0.05: \text{bold}, \text{best}: \text{bold} \]

4.3.1 Ripper if-then ruleset

The advantage of using high level musical features is that they can give useful insights in the characteristics of a composer’s style. A number of techniques are available to obtain a comprehensible model from these features. Rulesets and trees can be considered as the most easy to understand classification models due to their linguistic nature (Martens, 2008). Such models can be obtained by using rule induction and rule extraction techniques. The first category simply induces rules directly from the data, whereas rule extraction techniques attempt to extract rules from a trained black-box model (Martens et al., 2007). This research focuses on using rule induction techniques to build a ruleset and a decision tree.

In this section an inductive rule learning algorithm is used to learn “if-then” rules. Rulesets have been used in other research domains to gain insight in
credit scoring (Baesens et al., 2003), medical diagnosis (Kononenko, 2001), customer relationship management (Ngai et al., 2009), diagnosis of technical processes (Isermann and Balle, 1997) and more.

In order to build a ruleset for composer classification, the propositional rule learner RIPPER was used (Cohen, 1995). JRip is the Weka implementation of the “Repeated Incremental Pruning to Produce Error Reduction algorithm” (RIPPER). This algorithm uses sequential covering to generate the ruleset. It starts by learning one rule, removes the training instances that are covered by the rules, and then repeats this process (Hall et al., 2009).

Five rules were extracted by JRip, one for each composer, with 10-fold cross-validation. The rules are displayed in Figure 4.1. This figure shows that seven different features are used to decide if a piece is composed by Haydn, Bach or Beethoven. The “Most common melodic interval prevalence”, or the occurrence frequency of the interval that is most used, is present in most of the rules. This indicates that, for instance, Beethoven typically does not focus on using one particular interval, in contrast to Haydn or Bach, for whom the prevalence of the most common melodic interval is not as restrictive.

A good learning algorithm should be able to accurately predict new samples that are not in the training set. The accuracy of classification and AUC with 10-fold cross-validation are displayed in Table 4.3. The confusion matrix is displayed in Table 4.4. The latter table shows that least confusion occurs between Bach and Beethoven. The relatively higher misclassification rate between Haydn and Beethoven; and Haydn and Bach could be due to the fact that the dataset was larger for Bach and Haydn. A second reason could be that Haydn and Beethoven’s styles are indeed more similar, as suggested by the greater amount of chronological and geographical overlap between their lives, and by the fact that Haydn was once Beethoven’s teacher (DeNora, 1997). The timeline in Figure 4.2 shows they lived more in the same time period, just like Haydn and Bach. As for geographical proximity, Bach spent most of his life in North-East Germany (Leipzig, Köthen, Weimar) compared to Beethoven who moved from West-Germany (Bohn, Cologne) to Austria (Vienna), Haydn’s home country (Greene, 1985). While running the algorithm, the minimum total weight of the instances in a rule was deliberately set high
if (Most Common Melodic Interval Prevalence) ≤ 0.2688 and (Melodic Octaves Frequency ≥ 0.06399) then
    Composer = BE
else if (Most Common Melodic Interval Prevalence ≤ 0.2823) and (Most Common Pitch Prevalence ≤ 0.07051) and (Melodic Octaves Frequency ≥ 0.02489) and (Relative Strength of Top Pitches ≤ 0.9754) then
    Composer = BE
else if (Most Common Melodic Interval Prevalence ≤ 0.328) and (Repeated Notes Frequency ≥ 0.07592) and (Most Common Pitch Prevalence ≤ 0.1076) then
    Composer = HA
else if (Stepwise Motion Frequency ≤ 0.5732) and (Chromatic Motion Frequency ≥ 0.1166) and (Repeated Notes Frequency ≥ 0.3007) then
    Composer = HA
else
    Composer = BA
end if

Figure 4.1.: Ruleset

in order to get a smaller and thus more comprehensible tree, albeit slightly less accurate. Still overall, with 77% correctly classified and an AUC of 78%, the model developed with RIPPER is reasonably good.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>classified as</th>
</tr>
</thead>
<tbody>
<tr>
<td>145</td>
<td>42</td>
<td>67</td>
<td>a = HA</td>
</tr>
<tr>
<td>41</td>
<td>110</td>
<td>45</td>
<td>b = BE</td>
</tr>
<tr>
<td>25</td>
<td>17</td>
<td>553</td>
<td>c = BA</td>
</tr>
</tbody>
</table>
4.3. Composer Classification Models

Figure 4.2.: Timeline of the composers Bach, Beethoven and Haydn (Greene, 1985)

4.3.2 C4.5 Decision Tree

A second, tree-based, model is induced to get a more visual understanding of the classification process. Weka’s J48 algorithm (Witten and Frank, 2005) is used to build a decision tree with the C4.5 algorithm (Quinlan, 1993).

A decision tree is a tree data structure that consists of decision nodes and leaves. The leaves specify the class value, in this case the composer, and the nodes specify a test of one of the features. A path from the root to a leaf of the tree can be followed based on the feature values of the particular musical piece and corresponds to a predictive rule. The class at the resulting leaf indicates the predicted composer (Ruggieri, 2002). Although decision trees do not always offer the most accurate classification results, they are often useful to get an understanding of how the classification is done.

Similar to rulesets, decision trees have been applied to a broad range of topics such as medical diagnosis (Wolberg and Mangasarian, 1990), credit scoring (Hand and Henley, 1997), estimation of toxic hazards (Cramer et al., 1976), land cover mapping (Friedl and Brodley, 1997), predicting customer behaviour changes (Kim et al., 2005) and others. Much like rulesets, one of the main advantages of a decision tree model is its comprehensibility (Craven and Shavlik, 1996).
Unlike the covering algorithm implemented in the previous model, C4.5 builds trees recursively with a “divide and conquer” approach (Quinlan, 1993). This type of approach works from the top down, seeking a feature that best separates the classes, after which the tree is pruned from the leaves to the root (Wu et al., 2008).

The resulting decision tree is displayed in Figure 4.3. All four features from this tree model also occur in the ruleset (Figure 4.3). It is noticeable that the
feature evaluated at the root of the tree is the same feature that occurs in many of the rules from the if-then ruleset (see Figure 4.1). The importance of the “Most common melodic interval prevalence” feature for composer recognition is again confirmed, as it is the root node of the tree model. The “melodic octaves frequency” feature indicates that Bach uses more octaves than Haydn. Bach also seems to use less repeated notes.

Table 4.3 shows that the accuracy (79%) and AUC (83%) values of the tree are very comparable to those of the if-then rules extracted in the previous section. Again, the comprehensibility of the model was favoured above accuracy. Therefore, the minimum number of instances per leaf was kept high. The confusion matrix (see Table 4.5) is also comparable, with most classification errors occurring between Haydn and Beethoven. The least confusion can be seen between Beethoven and Bach, as in the previous model.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>classified as</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>174</td>
<td>31</td>
<td>49</td>
<td>a = HA</td>
</tr>
<tr>
<td></td>
<td>63</td>
<td>106</td>
<td>27</td>
<td>b = BE</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>14</td>
<td>541</td>
<td>c = BA</td>
</tr>
</tbody>
</table>

4.3.3 Logistic regression

In the previous sections, two comprehensible models are developed. These models provide crisp classification, which means that they determine if a musical piece is either composed by a certain composer or not. They do not offer a continuous measure that indicates “how much” characteristics of a certain composer are in a piece. In this section, a scoring model is developed that can accurately describe how well a musical piece belongs to a composer’s style.

Weka’s SimpleLogistic function was used to build a logistic regression model (Witten and Frank, 2005), which was fitted using LogitBoost. The
LogitBoost algorithm performs additive logistic regression (Witten and Frank, 2005). Boosting algorithms like LogitBoost sequentially apply a classification algorithm, a simple regression function in this case, to reweighted versions of training data. For many classifiers this simple boosting strategy results in dramatic performance improvements (Friedman et al., 2000).

A logistic regression model was chosen because it can indicate the statistical probability that a piece is written by a certain composer. This is useful for the inclusion in the objective function of the music generation algorithm, as described in the next section. Logistic regression models are less prone to overfitting than other models such as neural networks and require limited computing power (Tu, 1996). Logistic regression models can again be found in many areas, including the creation of habitat models for animals (Pearce and Ferrier, 2000), medical diagnosis (Kurt et al., 2008), credit scoring (Wiginton, 1980) and others.

The resulting logistic regression model for composer recognition is displayed in Equations 4.1 to 4.4. \( f_{\text{comp}}(i) \) represents the probability that a piece is composed by composer \( i \). This probability follows a logistic curve, as displayed in Figure 4.4. The advantage of using this model is that it outputs a number in the interval \([0,1]\), which can easily be integrated in the objective function of the music generation algorithm to assess how well a fragment fits into a certain composer’s style.

\[
\begin{align*}
    f_{\text{comp}}(L_i) &= \frac{1}{1 + e^{-L_i}} \\
    L_{HA} &= -3.39 + 21.19 \cdot x_1 + 3.96 \cdot x_2 + 6.22 \cdot x_3 + 6.29 \cdot x_4 - 4.9 \cdot x_5 \\
    &\quad - 1.39 \cdot x_6 + 3.29 \cdot x_7 - 0.17 \cdot x_8 + 0 \cdot x_9 \\
    &\quad - 0.72 \cdot x_{10} + 8.35 \cdot x_{11} - 4.21 \cdot x_{12}
\end{align*}
\]
4.3. Composer Classification Models

\[ L_{BE} = 6.19 + 5.44 \cdot x_1 + 14.69 \cdot x_2 + 24.36 \cdot x_3 - 0.45 \cdot x_4 - 6.52 \cdot x_5 \\
- 29.99 \cdot x_6 + 3.84 \cdot x_7 - 0.38 \cdot x_8 - 3.39 \cdot x_9 \\
- 2.76 \cdot x_{10} + 2.04 \cdot x_{11} - 0.48 \cdot x_{12} \]  

(4.3)

\[ L_{BA} = -4.88 - 13.15 \cdot x_1 - 6.16 \cdot x_2 - 5.28 \cdot x_3 - 11.63 \cdot x_4 + 11.92 \cdot x_5 \\
+ 34 \cdot x_6 - 13.21 \cdot x_7 + 3.1 \cdot x_8 + 2.37 \cdot x_9 \\
+ 0.66 \cdot x_{10} - 5.05 \cdot x_{11} + 3.03 \cdot x_{12} \]  

(4.4)

A musical piece is classified as being composed by composer \( i \) when it has the highest probability for that specific composer according to Equation 4.1 compared to the probabilities for other composers. A coefficient with a high absolute value indicates a feature that is important for distinguishing a particular composer. For example, \( x_5 \) (most common melodic interval frequency) has a high coefficient value, especially for BA. This feature is also at the top of the decision tree (Figure 4.3) and occurs in almost all of the rules from the ruleset (Figure 4.1).

With 83% correctly classified instances from the test set and an AUC value of 92%, the logistic regression model outperforms the previous models (see Table 4.3). This higher prediction accuracy is reflected in the confusion matrix (see Table 4.6). The average probability of the misclassified pieces is 64%. Examples of misclassified pieces include the Brandenburg Concerto No. 5 in D major, BWV 1050, Mvmt. 1 from Bach, which is classified as Haydn with a probability of 37% and String Quartet No. 9 in C major, Op. 59, No. 3, Allegro molto from Beethoven, which is classified as Haydn with a probability of 4%.
In this section, LibSVM was used to build a support vector machine (SVM) classifier (Chang and Lin, 2011). The support vector machine is a learning procedure based on the statistical learning theory (Vapnik, 1995). This method has been applied successfully in many areas including stock market prediction (Huang et al., 2005), text classification (Tong and Koller, 2002), gene selection (Guyon et al., 2002) and others. Given a training set of $N$ data points $\{(x_i, y_i)\}_{i=1}^{N}$ with input data $x_i \in \mathbb{R}^n$ and corresponding binary class labels $y_i \in \{-1, +1\}$, the SVM classifier should fulfil the following conditions (Cristianini and Shawe-Taylor, 2000; Vapnik, 1995):

\[
\begin{cases}
    w^T \phi(x_i) + b \geq +1, & \text{if } y_i = +1 \\
    w^T \phi(x_i) + b \leq -1, & \text{if } y_i = -1
\end{cases}
\]  

(4.5)
The non-linear function $\varphi(\cdot)$ maps the input space to a high (possibly infinite) dimensional feature space. In this feature space, the above inequalities basically construct a hyperplane $w^T \varphi(x) + b = 0$ discriminating between the two classes (see Figure 4.5). By minimizing $w^T w$, the margin between both classes is maximized. For more details on how this is done, the reader is referred to Chapter 7.

![Figure 4.5: Illustration of SVM optimization of the margin in the feature space.](image)

In this research, the Radial Basis Function (RBF) kernel (see Section 7.5.5) was used to map the feature space to a hyperplane. A hyperparameter optimization procedure was conducted with GridSearch in Weka to determine the optimal setting for the regularization parameter $C$ ($0.0001, 0.001, \ldots, 10,000$) and the $\sigma$ for the RBF kernel ($\sigma = 0.1, 1, \ldots, 100,000$). The choice of hyperparameters to test was inspired by settings suggesting by Weka (2013b). The Weka implementation of GridSearch performs 2-fold cross validation on the initial grid. This grid is determined by the two input parameters ($C$ and $\sigma$ for the RBF kernel). 10-fold cross validation is then performed on the best point of the grid based on the weighted AUC by class size and its adjacent points. If a better pair is found, the procedure is repeated on its neighbours until no better pair is found or the border of the grid is reached (Weka, 2013a).
The SVM classifier with non-linear kernel is a complex, non-linear function. Trying to comprehend the logic of the classifications made is quite difficult, if not impossible (Martens et al., 2009; Martens and Provost, 2014). The resulting accuracy is 86% and the AUC-value is 93% for the SVM with RBF kernel (see Table 4.3). The confusion matrix (Table 4.7) confirms that SVM is the best model for classifying between Haydn, Beethoven and Bach. Most misclassification occurs between Haydn and Beethoven, which can be explained by the geographical and temporal overlap between the lives of these composers as mentioned in Section 4.3.1.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>classified as</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>26</td>
<td>24</td>
<td>a = HA</td>
</tr>
<tr>
<td>49</td>
<td>127</td>
<td>20</td>
<td>b = BE</td>
</tr>
<tr>
<td>22</td>
<td>10</td>
<td>563</td>
<td>c = BA</td>
</tr>
</tbody>
</table>

The ROC curves of the two best models according to Table 4.3 are displayed in Figure 4.6. The ROC curve displays the trade-off between true positive rate (TPR) and false negative rate (FNR). Both models clearly score better than a random classification, which is represented by the diagonal through the origin. Although both models have a high AUC value, the ROC curves for the SVM score slightly better. When examining the misclassified pieces, they all seem to have a very low probability, with an average of 39%. Examples of misclassified pieces are String Quartet in C major, Op. 74, No. 1, Allegro moderato from Haydn, which is classified as Bach with 38% probability and Six Variations on a Swiss Song, WO 64 (Theme) from Beethoven, which is classified as Bach with a probability of 38%.

### 4.4 Generating Composer-Specific Music

In order to generate music with composer-specific characteristics, the existing objective function from Chapter 2 for evaluating counterpoint ($f_{cp}$) was
Figure 4.6.: ROC curves of the best performing models

extended with the probabilities of the logistic regression model. This model was preferred over the slightly more accurate SVM model since it is easy to comprehend (Martens et al., 2007) and returns a clearly defined probability per composer. The resulting objective function for composer $i$ is displayed in Equation 4.6. An approach related to this research, yet using different classes and features, is described by Pachet (2009). Global features are used to develop a support vector machine classification model (SVM) that can classify between tonal, brown, serial, long and short melodies. Existing melodies were then transformed into another type by improving the SVM’s score for this particular melody. When composing with characteristics of a certain composer $i$, the weights $a_i$ should be set high and the others to 0. This ensures that only the counterpoint characteristics and those of composer $i$ are taken into account. A low score corresponds to better contrapuntal music with more influences of composer $i$.

$$f_i = f_{cp} + \sum_{i \in BE,BA,HA} a_i \cdot (1 - f_{comp}(L_i)) \quad (4.6)$$
The new model was added to the existing objective function for counterpoint in order to ensure that some basic harmonic and melodic rules are still checked. By temporarily removing the first term from Equation 4.6, it is quickly confirmed by listening that generating music that only adheres to the rules extracted in the previous section, without optimizing $f_{cp}$, does not result in musically meaningful results. The counterpoint rules are therefore necessary in order to ensure that the generated music also optimizes some basic musical properties such as “only consonant intervals are permitted”. With this new objective function, the VNS algorithm is able to generate contrapuntal music with characteristics of a certain composer.

4.4.1 Implementation - FuX

The Android app described in Chapter 3 was extended to include this VNS implementation. The graphical user interface of FuX 2.0 is displayed in Figure 4.7. The three sliders (or SeekBars) give the user control over the weights $a_i$ of the objective function (see Equation 4.6). This even allows a user to generate music consisting of a mix of multiple composers if he or she wishes to do so. The playback instrument can be chosen and dynamically changed by the user.

4.4.2 Results

The resulting composer-specific music generation algorithm was tested on an Eclipse Android Virtual Device with Android 4.0.3, ARM processor and 512MB RAM. The emulator was installed on an OpenSuse system with Intel®Core™2 Duo CPU@ 2.20GHz and 3.8GB RAM. Figure 4.8 displays the evolution of the solution quality of the best found solution over time. The discrete points are connected by lines for visual clarity. The left plots describe the generation of the cantus firmus (CF) or bass line. The plots on the right hand side describe the evolution of the score of the counterpoint (CP) line or top line. In the experiment 16 measures are generated with a cutoff time of
12 seconds. FuX first generates the bass line and continues with the top line. Since the main objective is to produce music with composer-specific characteristics, the weight of the respective composer’s score is set very high (100), this ensures that the probability of a composer is preferred by the algorithm over the counterpoint rules. Figure 4.8 shows a drastic improvement of the selected composer’s score for each of the three composers. For example, in Figure 4.8(a) $f_{comp}(L_{HA})$ goes down rapidly over time, while $f_{comp}(L_{BE})$ and $f_{comp}(L_{BA})$ remain relatively stable. This means that the generated music actually contains composer-specific elements according to the logistic regression model that was built. When optimizing for a specific composer no real change in the scores for the other composers can be noted. The improvements of the counterpoint score are not very high, but are enough to add basic musical properties to the fragment. Of course, the thought processes of the great composers are far more complex than can be captured by the melodic features used in this research. The limited set of composer-specific characteristics (see Table 4.2) that FuX overlays on the counterpoint fragment are not enough by themselves to generate a piece of music that would be recognized as being
composed by one of the selected composers. The generated music does however contain characteristics of the selected composer. All three composers used in this research composed music that differs in much more than the limited set of characteristics that FuX controls. Bach, e.g., worked in the Baroque period while Beethoven’s work was composed during the transition from the Classical to the Romantic Era. The differences in style between the musical styles common during these different periods are vastly more encompassing than simple variations in the melodic characteristics recognized by FuX. This research can be seen as a first step towards creating a system that is able to generate more complete musical pieces in the style of a certain composer.

Due to large differences in computing power between different Android devices, the quality of the generated music is highly dependent on the architecture of the mobile device on which it is run. Still, the subjective opinion of the authors is that the generated stream of music sounds pleasant to the ear, even on relatively modest hardware. The reader is invited to install the app and listen to the resulting music. Some examples of generated music, including their probability per composer according to the logistic regression model, are given in Figures 4.9, 4.10 and 4.11.

4.5 CONCLUSIONS

A number of musical features were extracted from a large database of music. Based on these features four classification models models were built. The first two models, an if-then ruleset and a decision tree, give the user more insight and understanding in the musical style of a composer, e.g., “Beethoven typically does not focus on using one particular interval, in contrast to Haydn or Bach, who have a higher prevalence of the most common melodic interval”. The other two models, a logistic regression and a support vector machine classifier, can more accurately classify musical pieces from Haydn, Beethoven and Bach. The first of these models is integrated in the objective function of a variable neighbourhood search algorithm that can efficiently generate contrapuntal music. The resulting algorithm was implemented as a user friendly Android app called FuX, which is able to play a stream of contrapuntal
Figure 4.8.: Evolution of solution quality over time

\[
\begin{align*}
\text{(a) Haydn (CF)} & : \quad f_{\text{comp}}(L_{HA}) \quad \text{(dotted line)} \\
\text{(b) Haydn (CP)} & : \quad f_{\text{comp}}(L_{BE}) \\
\text{(c) Bach (CF)} & : \quad f_{\text{comp}}(L_{BA}) \\
\text{(d) Bach (CP)} & : \quad f_{\text{comp}}(L_{HA}) \\
\text{(e) Beethoven (CF)} & : \quad f_{\text{comp}}(L_{BE}) \\
\text{(f) Beethoven (CP)} & : \quad f_{\text{comp}}(L_{BA})
\end{align*}
\]
music with composer-specific characteristics that sounds pleasing, at least to the subjective ear of the authors.

Combining a certain composer’s characteristics with the counterpoint style creates a peculiar fusion of styles, yet this approach is merely an initial step towards a more complete system. In the future it would be interesting to work with composer classification models built on a dataset of one particular style (e.g., string quartets). Enforcing basic musical properties while generating pieces with characteristics specific to a composer might then be done by integrating rules specific to the chosen style instead of the currently used counterpoint rules. Other future extensions of this research include working with other musical styles, voices and adding a recurring theme to the music. This will be examined further in the next chapters, by generating music from statistical models of other styles and by taking into account a larger structure.
4.5. CONCLUSIONS

Figure 4.10.: A generated fragment with high probability for Beethoven (99.64%)

Figure 4.11.: A generated fragment with high probability for Haydn (99.31%)
This chapter is based on the paper “D. Herremans, K. Sørensen and D. Conklin. 2014. Sampling the extrema from statistical models of music with neighbourhood search. Proceedings of ICMC–SMC. Athens. 1096–1103.”
To evaluate musical quality, the variable neighbourhood search algorithm used predefined style rules from music theory combined with a learned model that imposed characteristics of a certain composer in the previous chapter. In this chapter it has been made more versatile by evaluating music based only on an automatically learned model, while maintaining its efficiency. The vertical viewpoints method is used to learn a Markov model of abstract features from a corpus of first species counterpoint. This model is incorporated in the objective function of the variable neighbourhood search algorithm. There are several different techniques to generate music from a statistical model, but not all are able to effectively explore the higher probability extrema of the distribution of sequences. The resulting system is extensively tested and compared to other popular sampling algorithms such as Gibbs sampling and random walk.

5.1 INTRODUCTION

To circumvent the human fitness bottleneck, most systems automatically assess the quality of a musical fragment. This can be done based on existing rules from music theory or by learning from a corpus of existing musical pieces. The first strategy has been applied in automatic composition systems such as those by Geis and Middendorf (2007) and Assayag et al. (1999). An obvious disadvantage is that the rules of the chosen musical style need to be formally written down.

Although every musical genre has its own rules, these are generally not explicitly available (Moore, 2001). Therefore, it is useful to automatically learn style rules from existing music. The second method can be considered as being more robust and expandable to other styles. David Cope’s Experiments in Musical Intelligence (EMI) extract signatures of musical pieces using pattern matching with a grammar based system to understand a specific composer’s style (Papadopoulos and Wiggins, 1999). Xenakis (1992) uses Markov models to control the order of musical sections in his composition “Analogique A”. Markov models have also been used to generate Palestrina counterpoint based on a given cantus firmus with dynamic programming (Farhood and Schoner,
Allan and Williams (2005) trained hidden Markov models for harmonising Bach Chorales and Whorley et al. (2013) applied a Markov model based on the multiple viewpoint method to generate four-part harmonisations. Markov models also form the basis for some real-time improvisation systems (Dubnov et al., 2003a; Pachet, 2003; Assayag and Dubnov, 2004) and more recent work on Markov constraints for generation (Pachet and Roy, 2011).

In this chapter we adopt the view that music generation can be viewed as sampling high probability sequences from statistical models of a music style. The question if high probability sequences offer the best musical quality is addressed in the next chapter. Although many systems are available to learn styles from existing music, few have been combined with an efficient optimization algorithm such as VNS. This is important since generating high probability sequences from complex statistical models containing multiple conditional dependencies between variables can be a computationally hard problem. In this research the vertical viewpoints method (Conklin, 2002) is applied to learn a model that quantifies how well music resembles first species counterpoint. This model is then used to replace the rule-based objective function in the VNS developed in the previous chapters.

We chose to work with simple first species counterpoint in this chapter in order to explore the theoretical concepts of sampling. It is not the goal of this research to develop a complete model, but evaluate the different methods to sample from a statistical model. The next section describes the statistical model used in this study and Section 5.3 describes the sampling methods used. The statistical model was chosen so that the optimal (Viterbi) solution could be computed, allowing us to evaluate the absolute in addition to the relative performance of various sampling methods. In Section 5.4 the resulting system is extensively tested and compared to the optimal solution and to the random walk and Gibbs sampling methods.
5.2 VERTICAL VIEWPOINTs

This section describes the model that provides the probabilities of each note in a first species counterpoint fragment. First species counterpoint can be viewed as a sequence of *dyads*, i.e., two simultaneous notes (see Figure 5.1). In this research the number of possible pitches is constrained to the scale of C major. The range of the cantus firmus, i.e., the fixed voice against which the counterpoint line is composed, is constrained to 48 and 65 (in MIDI pitch values) and the counterpoint ranges from 59 to 74. These constraints are based on counterpoint examples from Salzer and Schachter (1969). This results in 110 possible dyads (11 \times 10).

When generating counterpoint fragments, it is essential to consider both vertical (harmonic) and horizontal (melodic) aspects. These two dimensions should be linked instead of treated separately. Furthermore, in order to confront the data sparsity issue in any corpus, abstract representations should be used instead of surface representations. These representational issues are handled by defining a *viewpoint*, a function that transforms a concrete event into an abstract feature. In this chapter the *vertical viewpoints* method (Conklin, 2002; Conklin and Bergeron, 2010) is used to model harmonic and melodic aspects of counterpoint.

![Figure 5.1: First species counterpoint example (Salzer and Schachter, 1969) and its dyad representation](image)

A simple linked viewpoint is used whereby every dyad is represented by three linked features (see Figure 5.2): two melodic pitch class intervals
between the two melodic lines, and a vertical pitch class interval within the dyad. With this representation, the second dyad $b$ in Figure 5.2 is given by the compound feature $\tau(b \mid a) = (2, 5, 3)$.

Following this transformation, dyad sequences in a corpus are transformed to more general feature sequences, which are less sparse than the concrete dyad sequence for obtaining statistics from a corpus. In the following it is described how to create a simple first order transition matrix (TM) over dyads from these statistics, which can immediately be applied in any optimization algorithm (see Section 5.3.1).

Following the method of Conklin (2013b), let $v = \tau(b \mid a)$ be the feature assigned by a viewpoint $\tau$ to dyad $b$, in the context of the preceding dyad
Assuming the probabilistic graphical model of Figure 5.3, the probability $P(b \mid a)$ of dyad $b$ following dyad $a$ can be derived as follows:

$$P(b \mid a) = P(a,b)/P(a) \quad \text{conditional probability}$$
$$= P(b,v,a)/P(a) \quad \text{because } P(v \mid b,a) = 1$$
$$= P(b \mid v,a) \times P(v,a)/P(a) \quad \text{chain rule}$$
$$= P(b \mid a,v) \times P(v) \quad \text{independence of } a \text{ and } v$$

with the second term $P(v)$ estimated from the corpus:

$$P(v) = c(v)/n$$

where $n$ is the number of dyads in the corpus and $c(v)$ is the number of dyads in the corpus having the feature $v$. To further reduce the number of parameters for training simply to the quantities $P(v)$, the first term $P(b \mid a,v)$ is modelled with a uniform distribution

$$P(b \mid a,v) = |\{x : \tau(x \mid a) = v\}|^{-1}$$

where $x$ ranges over all 110 possible dyads.

As an example of the calculation of $P(b \mid a)$, referring to the first two dyads of Figure 5.2, consider the probability of the second dyad $b = [65, 50]$ following the first dyad $a = [60, 48]$. Suppose that $P(⟨2, 5, 3⟩) = 0.01687$. Given the space of possible dyads, we have

$$|\{x : \tau(x \mid [60, 48]) = ⟨2, 5, 3⟩\}| = 6$$

that is, there are 6 possible dyads

$$\{[65, 48], [65, 50], [65, 62], [60, 48], [60, 50], [60, 62]\}$$

that have the feature $⟨2, 5, 3⟩$ in the context of dyad $[60, 48]$. Therefore for this example:

$$P([65, 50] \mid [60, 48]) = 1/6 \times 0.01687 = 0.0028$$
A complete statistical model is created by filling a transition matrix of dimension $110 \times 110$ with these quantities for all possible pairs of dyads.

Given a first order transition matrix over dyads, the probability $P(s)$ of a sequence $s = e_1, \ldots, e_\ell$ consisting of a sequence of $\ell$ dyads is given by

$$P(s) = \prod_{i=2}^{\ell} P(e_i \mid e_{i-1})$$  \hspace{1cm} (5.1)

This probability will be used to create an objective function, as discussed in the following section.

5.3 Sampling High Probability Solutions

In this research generating counterpoint music is seen as a combinatorial optimization problem, whereby the best combination of notes needs to be found in order to produce music that adheres to a certain style as well as possible. Since generating dyad sequences with the best possible objective function is a computationally hard problem, a variable neighbourhood search algorithm (VNS) is used as it is an efficient optimization method.

The VNS previously developed in Chapter 1 is adapted to work with a learned objective function. The VNS method is then compared with two sampling methods i.e., random walk, and Gibbs sampling. One of the reasons for using a first order Markov model to represent first species counterpoint is that it is possible to compute the Viterbi optimum for this problem. This allows a thorough comparison of the sampling methods relative to each other and the optimum solution (see Section 5.4.2).

5.3.1 Objective Function

In Section 5.2 a Markov model with vertical viewpoints was described for learning the characteristics of a corpus of musical pieces. This statistical
model is transformed into an objective function that can be used to indicate the quality of a generated fragment. High solution quality corresponds to high probability (Equation 5.1) sequences in the model. High probability sequences will guarantee a sequence of the most likely transitions. Whether this necessarily corresponds to a good sounding musical piece in its entirety will be examined in more detail in the next chapter.

The probability $P(s)$ is transformed into cross-entropy since it is more convenient to use logarithms. The sum of the logarithms is normalised by the sequence length to obtain the cross-entropy $f(s)$:

$$f(s) = -\frac{1}{\ell-1} \sum_{i=2}^{\ell} \log_2 P(e_i \mid e_{i-1})$$  \hspace{1cm} (5.2)

The quality of a counterpoint fragment is thus evaluated according to the cross-entropy (average negative log probability) of the fragment computed using the dyad transitions of the transition matrix. This forms the objective function $f(s)$ that should be minimized.

The Viterbi solution, the minimal cross-entropy solution, can be computed directly from the transition matrix. This is done by a dynamic programming algorithm, which fills a solution matrix of dimension $110 \times \ell$ columnwise, accumulating the best partial path ending at each dyad and sequence position in each cell. The minimal cross-entropy is given by the minimum value within the last column of the solution matrix.

5.3.2 Variable neighbourhood search

The structure of the implemented VNS is represented in Figure 5.4. The components are mostly the same as described in Section 1.3. The stopping criterion is based on the number of lookups in the transition matrix (TMLookups). The
5.3. Sampling High Probability Solutions

VNS algorithm was implemented in C++ and the source code is available online\(^1\).

---

\(^1\) [http://antor.ua.ac.be/musicvns](http://antor.ua.ac.be/musicvns)
5.3.3 Random walk

The random walk method (Conklin, 2003) is a simple and common way to generate a sequence from a Markov model. The initial dyad is fixed (see Section 5.4). After that, successive dyads are generated by sampling from the probability distribution given by the relevant row of the transition matrix (based on the previous dyad). That is, at each position $i$ the next dyad $e_i$ is selected with probability $P(e_i | e_{i-1})$. If there is no next dyad with non-zero probability, the entire process is restarted from the beginning of the sequence. Iterated multiple times, on every successful iteration, the cross-entropy of the solution is noted if it is better than the best score so far.

5.3.4 Gibbs sampling

Gibbs sampling is a popular method used in a wide variety of statistical problems for generating random variables from a (marginal) distribution indirectly, without having to calculate the density (Casella and George, 1992). The algorithm is given a random piece $s$ generated by the random walk method above. The following process is iterated: a random location in the piece $s$ is chosen and all valid dyads are substituted into that position, each substitution producing a new piece $s'$ having probability $P(s')$. This distribution over all modified pieces is normalized, and one is sampled from this distribution. This process is iterated with $s$ set to the sampled piece. Iterated multiple times, on every iteration the cross-entropy of the solution is noted if it is better than the best score so far.

5.4 Experiment

In order to compare the efficiency of the VNS with other techniques an experiment was set up. Since there are no large available corpora restricted to first species counterpoint, 1000 pieces were generated by means of the
algorithm with a rule-based objective function from Chapter 1 (Herremans and Sörensen, 2012). All pieces consist of 64 dyads. These pieces were used to train the Markov model discussed in section 5.2.

A number of hard constraints are imposed to better define and limit the problem. Firstly, as discussed in Section 5.2, the range is restricted to 110 dyads. Secondly, the cantus firmus is specified and cannot be changed by the algorithm (thus, the three optimization methods in Section 5.3 consider only those dyads compatible with the specified cantus firmus). Based on music theory rules specified by Salzer and Schachter (1969), a third hard constraint fixes the first dyad to [60 48] and the last two dyads to [59 50] and [60 48]. This brings the number of possible solutions to $10^{61}$. The Viterbi solution for this problem has a cross-entropy of 3.22410 (see Table 5.1).

5.4.1 Distribution of random walk

Figure 5.5 shows the distribution of cross-entropy of musical sequences sampled by random walk. A total of $10^7$ iterations of random walk sampling were performed, and the cross-entropies (excluding those solutions which led to a dead end during the random walk) were plotted. The plot therefore shows the probability of random walk producing some solution within the indicated cross-entropy bin. The difficulty of sampling high probability solutions is immediately apparent from the graph. For example, in order to generate one solution in the cross-entropy range of 3.5 – 3.6 (a solution still worse than the Viterbi solution), approximately one million solutions should be sampled with random walk. Figure 5.5 also shows that even with the large number of random walk samples taken ($10^7$), the Viterbi solution is not found.

5.4.2 Performance of the sampling algorithms

To evaluate the relative performance of the sampling methods, the number of transition matrix lookups (TM lookups) is used as a complexity measure
Figure 5.5.: The distribution of cross-entropy according to random walk, over $10^7$ iterations

Table 5.1.: Average and best results of 100 runs after 30 million transition matrix lookups

<table>
<thead>
<tr>
<th></th>
<th>Viterbi</th>
<th>random walk</th>
<th>Gibbs sampling</th>
<th>VNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>average</td>
<td>n/a</td>
<td>3.64788</td>
<td>3.53426</td>
<td>3.24453</td>
</tr>
<tr>
<td>best</td>
<td>3.22410</td>
<td>3.44837</td>
<td>3.39871</td>
<td>3.22410</td>
</tr>
</tbody>
</table>

in order to compare the VNS with random walk and Gibbs sampling. A total of 100 runs are performed with a cut-off point of $3 \times 10^7$ TM lookups or alternatively until the Viterbi solution is reached.

The average of the best scores of each of the 100 runs are displayed in Table 5.1 per algorithm. A one-sided Mann-Whitney-Wilcoxon test was performed to test if the results attained by the VNS are significantly better than the ones from the random walk and Gibbs sampling. Since the p-values of the latter algorithms are both $< 2.2 \times 10^{-16}$ we can accept the alternative hypothesis which states that the results from the VNS are lower (i.e., better) than the ones for both random walk (RW) and Gibbs sampling (GS). The VNS was able to
find the optimal fragment \( f(s) = 3.2241 \) before the cut-off point of \( 3 \times 10^7 \) TM lookups in 51% of the cases. Neither GS nor RW were able to reach the optimum in any of the iterations. The best cross-entropy values reached by all three of the algorithms during the 100 runs are displayed in Table 5.1.

Figure 5.6 shows the evolution of the average value of the objective function for the best fragment found by the algorithms over 100 runs. The ribbons on the graph indicate the best and worst run of each algorithm. The Viterbi optimum is displayed as the lower horizontal line. It is clear from the graph that VNS outperforms both GS and RW. All three algorithms seem to start with a very steep descent in the very beginning of the run, but GS and RW converge faster. Gibbs sampling does perform slightly better than random walk, but the best run is still worse than the worst run of the VNS.

Figure 5.7 focuses on the first 50,000 TM lookups displayed in Figure 5.6. In the very beginning of the runs, VNS is outperformed by the two simpler
algorithms. This is probably due to the fact that the VNS starts from a random initial solution that allows zero-probability transitions. Even so, the algorithm is able to quickly improve these solutions. A combination of VNS with an initial starting solution generated by a random walk could even further improve its efficiency.

Figure 5.7.: Evolution of the best fragment found zoomed in on the beginning of the runs

5.5 CONCLUSIONS

The approach used in this chapter shows the possibilities of combining music generation with machine learning and provides us with an efficient method to
generate music from styles whose rules are not documented in music theory. The proposed VNS algorithm is a valid and flexible sampling method that is able to find the fragment with highest probability dyad transitions according to a learned model. It outperforms both random walk and Gibbs sampling in terms of sampling of high probability solutions. The focus of this chapter is on high probability (low cross-entropy) regions, but the VNS can just as easily be applied to sample other regions such as low probability regions. In addition to the VNS contribution, in this chapter we confirmed that random walk does not practically (only in the theoretical limit of iterations) sample from the extrema (i.e., sampling the highest probability pieces), from even a simple Markov model.

It must be mentioned that the absolute cross-entropy results presented in this chapter possibly have some bias towards the VNS method, because the moves used to generate the training data for the creation of the statistical model are in fact the same as those used by VNS during the search of the solution space. Nevertheless, we expect the relative performances of the sampling methods to hold up under independent training data since the difference in performance is very large. In future research, other metaheuristics might be compared to the VNS on a learned corpus.

The results are promising as the VNS method converges to a good solution within relatively little computing time. The described VNS is a valid and flexible sampling method and has been successfully combined with the vertical viewpoints method. In future research, these methods will be applied to higher species counterpoint (Herremans and Sørensen, 2013; Whorley et al., 2013; Conklin and Bergeron, 2010) with the multiple viewpoint method (Conklin, 2013b; Conklin and Witten, 1995), using more complex learned statistical models. When generating more complex music, new move types should be added to the VNS in order to escape local optima. The consideration of more complex contrapuntal textures will also permit the use of a real corpus. In the next chapter, different methods for using the Markov models for quality assessment of structured music are compared and evaluated. The musical output and validity of high probability sequences versus other evaluation metrics are critically examined.
GENERATING STRUCTURED MUSIC USING QUALITY METRICS BASED ON MARKOV MODELS

This chapter is based on the paper “D. Herremans, S. Weisser, K. Sörensen and D. Conklin. 2014. Generating structured music using quality metrics based on Markov models. Working paper 2014019. Faculty of Applied Economics, University of Antwerp.”
In this chapter, a first order Markov model is built from a corpus of bagana music, a traditional lyre from Ethiopia. Different ways in which low order Markov models can be used to build quality assessment metrics for an optimization algorithm are explained. These are then implemented in a variable neighbourhood search algorithm that generates bagana music. The results are examined and thoroughly evaluated. Due to the size of many datasets it is often only possible to get rich and reliable statistics for low order models, yet these do not handle structure very well and their output is often very repetitive. A method is proposed that allows the enforcement of structure and repetition within music, thus handling long term coherence with a first order model.

6.1 INTRODUCTION

Music generation systems can be categorised into two main groups. On the one hand are the probabilistic methods (Allan and Williams, 2005; Conklin and Witten, 1995; Xenakis, 1992), and on the other hand are optimization methods such as constraint satisfaction (Truchet and Codognet, 2004) and metaheuristics such as evolutionary algorithms (Horner and Goldberg, 1991; Towsey et al., 2001), ant colony optimization (Geis and Middendorf, 2007) and variable neighbourhood search (VNS) (Herremans and Sorensen, 2013). The first group considers the solution space as a probability distribution, while the latter optimizes an objective function on a solution space. In this chapter, we aim to bridge the gap between those approaches that consider music generation as an optimization system and those that generate based on a statistical model.

The main challenge when using an optimization system to compose music is how to determine the quality of the generated music. Some systems let a human listener specify how “good” the solution is on each iteration (Horowitz, 1994). GenJam, a system that composes monophonic jazz fragments given a chord progression, uses this approach (Biles, 2003). This type of objective function considerably slows down the algorithms (Tokui and Iba, 2000) and is known in literature as the human fitness bottleneck.
Most automatic composition systems avoid this bottleneck by implementing an automatically calculated objective function based either existing rules from music theory or by learning from a corpus of existing music. The first strategy has been used in compositional systems such as those of Geis and Middendorf (2007); Assayag et al. (1999) and Herremans and Sørensen (2013). Although every musical genre has its own rules, these are usually not explicitly available, which poses huge limits on the applicability of this approach (Moore, 2001). This problem is overcome when style rules can be learned automatically from existing music. This approach is more robust and expandable to other styles.

Markov models have been applied in a musical context for a long time. The string quartet called the Illiac Suite was composed by Hiller and Isaacson in 1957 by using a rule based system that included probability distributions and Markov processes (Sandred et al., 2009). Pinkerton (1956) learned first order Markov models based on pitches from a corpus of 39 simple nursery rhyme melodies, and used them to generate new melodies using a random walk method. Fred and Carolyn Atteave generated two perfectly convincing cowboy songs by performing a backward random walk on a first order transition matrix (Cohen, 1962). Brooks et al. (1957) learned models up to order 8 from a corpus of 37 hymn tunes. A random process was used to synthesise new melodies from these models.

An interesting conclusion from this early work is that high order models tend to repeat a large part of the original corpus and that low order models seem very random. This conclusion was later supported by other researchers such as Moorer (1972), who states: “When higher order methods are used, we get back fragments of the pieces that were put in, even entire exact repetitions. When lower orders are used, we get little meaningful information out”. These conclusions are based on a heuristic method whereby the pitch is still chosen based on its probability, but only accepted or not based on several heuristics which filter out, for instance, long sequences of non-tonic chords that might otherwise sound dull. Music compositions systems based on Markov chains need to find a balance in the order to use.

Other music generation research with Markov includes the work of Tipei (1975), who integrates Markov models in a larger compositional model. Xe-

In this chapter, a first order Markov model is built that quantifies note transition probabilities from a corpus of bagana music, a traditional lyre from Ethiopia. This model is then used to evaluate music with a certain repetition structure, generated by the VNS developed in the previous chapters (Herremans and Sørensen, 2012). Due to the size of many available corpora of music, including the bagana corpus used in this chapter, rich and reliable statistics are often only available for low order Markov models. Since these models do not handle structure and can produce very repetitive output, a method is proposed for handling long term coherence with a first order model. This method will also allow us to efficiently calculate the objective function, by using the minimal number of necessary note intervals as possible while still containing all information about the piece. Secondly, this chapter will critically evaluate how Markov models can be used to construct evaluation metrics in an optimization context. In the next section more information is given about bagana music, followed by an explanation of the technique employed to generate repeated and cyclic patterns. An overview of the different methods by which a Markov model can be converted into an objective function are discussed in Section 6.3. Variable neighbourhood search, the optimization method used to generate bagana music, is then explained. An experiment is set up and the different evaluation metrics are compared in Section 6.5.
6.2 STRUCTURE AND REPETITION IN BAGANA MUSIC

Bagana is a ten-stringed box-lyre played by the Amhara, inhabitants of the Central and Northern part of Ethiopia. It is an intimate instrument, only accompanied by a singing voice, which is used to perform spiritual music. It is the only melodic instrument played exclusively for religious purposes (Weisser, 2012). The bagana melody and singing voice are quasi homophonic, meaning that the voice and bagana usually follow each other in unison (Weisser, 2005). In this chapter the focus is on analysing and generating the instrumental part.

The bagana is made of wooden pillars and soundbox, equipped with ten cattle gut strings. The strings are plucked with the left hand and four strings are used as finger rests. It is tuned to an Amhara traditional pentatonic scale. Each finger of the left hand is assigned to one string (see Figure 6.1), except in the case of the index finger (referred to as finger 2 and 2′ in the figure), which plays two equally tuned strings. This allows us to make abstraction from the actual pitch and work with the corpus made by Conklin and Weisser (2014) based on finger numbers (see Section 6.5).

Bagana songs are typically very repetitive with a very recognisable overall structure (Weisser, 2006). This repetition is intentional since repetitive music has a strong influence on the state of consciousness among musical traditions. Even Western-trained listeners describe the sounds as “becoming meditative objects, relaxing the mind” (Dennis, 1974).

An example bagana song, including finger numberings, is given in Figure 6.2. Note that this piece consists of two sections, and that only a few segments (A¹, A² and A³) are used, and repeated many times throughout the duration of the song. Additionally, note that the segment A² appears within different sections of the piece. In what follows, an approach is described for respecting this structure and repetition within new sequences generated from Markov models.

Since repetition is so important for bagana music, cycles and repetitions must be represented and evaluated in an efficient way. Markov models alone are
incompatible of representing such structures, which can involve arbitrarily long-range dependencies, and therefore the approach used here is to preserve the structure and repetition provided by an existing template piece. The next subsections will describe a method for representing and efficiently evaluating this structure and repetition while still employing a Markov model to generate the basic musical material.

6.2.1 Cycles and patterns

Following the theoretical approach of Angluin (1980), the structure of a bagana piece may be represented using a pattern, which is a sequence of variables drawn from a set $V$ (we use $A^1, A^2, \ldots$ as variables). Given a set $\xi$ of event symbols (in the case of bagana, finger numbers), a realization of a pattern is a substitution from $V$ to $\xi^*$ (the set of all sequences formed from event symbols), mapping variables to sequences of finger numbers. Each variable is
6.2. STRUCTURE AND REPLICATION IN BAGANA MUSIC

Figure 6.2: Tew Semagn Hagere by Alemu Aga, as transcribed by Weisser (2005)

Also associated with a length, that is, a constraint on the length of the sequence that can replace the variable. The event sequence replacing a variable $A^i$, associated with a length $e$, will be notated in this chapter as $a^i_1a^i_2\ldots a^i_e$.

To represent repetition of entire sections, the notion of cycles and cyclic patterns is introduced. A cycle is a sequence of events that is repeated any number of times. For example, in the bagana song of Figure 6.2, the two cycles are the event sequences labelled by $A^1A^2$ and $A^3A^2$. Cycles can be abstracted and represented as cyclic patterns, which are patterns as described above but now enclosed in the symbols $\|:\|$ and $:\|\|$. For example, in the bagana song of Figure 6.2, the two cyclic patterns are $\|:A^1A^2:\|$ and $\|:A^3A^2:\|$. Patterns can also be concatenated, forming compound patterns. Taking the bagana song of Figure 6.2 as an example, the pattern describing this piece is finally represented as the compound pattern:

\[\|:A^1A^2:\|\|:A^3A^2:\|\] (6.1)

with the lengths of $A^1$, $A^2$, $A^3$ being specified as 6, 6, and 13, respectively. The corpus used in this research (see Section 6.5.1) has been annotated with these repetition structures by a bagana expert.

6.2.2 Realizing and evaluating cyclic patterns

A realization of a pattern is a mapping from variables of the pattern onto actual events (i.e., finger numbers). The events represented by any one variable are generated using a Markov model and the entire generation is given by
replicating the instances of the same variable. In order to properly generate music that contains cyclic patterns, traditional statistical sampling methods like random walk are not suited because long-range dependencies cannot be captured by a low order Markov process. Therefore, we use a local search optimization technique to generate the variables in this research. The actual realizations of the events are given to the objective function in order to assess the quality of a generated fragment.

In order to reduce the number of transition matrix lookups, without losing any information about the sequence, an expansion technique was developed to generate the minimal extended subsequence that can be used to calculate the objective function. For example, consider a cycle \( A = a_1 a_2 \ldots a_k \) that is repeated \( n \) times in the template piece. When calculating the objective function, we should take care not to omit the sequence \( a_k a_1 \), which is the transition that is heard whenever the cycle is repeated. Since calculating the objective function on \( A \) alone is not sufficient, we could simply calculate it on the full sequence as it is played, but this would require roughly \( n \) times more transition matrix lookups than required. The expanded sequence \( A' \) will simply contain an additional element, which represents the transition from end to beginning: \( A' = a_1 a_2 \ldots a_k a_1 \). The expansion method used in this research reduces the number of lookups while retaining all the information of individual transitions.

### 6.2.3 Compound cyclic patterns

Bagana music is characterised by a large number of repetitions combined together. The expansion method discussed in the previous subsection is applied to reduce the number of transition matrix lookups. This method keeps the minimum number of intervals without forgetting the connections between the end and beginning of a cycle, as discussed in the subsection above. For a compound pattern which contains cycles, some care needs to be taken to exclude certain intervals. For example, for the cyclic pattern described by Equation 6.1, the sequence on which the objective function is calculated thus becomes:
whereby $A^i$ consists of the note sequence $a^i_1 a^i_2 \ldots a^i_e$ and the $\downarrow$ represent discontinued intervals which should be excluded from the calculation.

This method as described above is valid for first order evaluation. When an evaluation metric is based on note sequences of more than two subsequent notes (e.g., unwords, see Section 6.3.5), higher order expansion is necessary. In the case of a metric that evaluates sequences of length 3, second order expansion is necessary, and the expanded sequence becomes:

$$A^1 A^2 a^1_1 \downarrow a^2_e A^3 A^2 a^3_1$$

where as before the $\downarrow$ represents a discontinued interval. In the next section, different methods of using of Markov models to construct quality metrics for an optimization algorithm are explained.
between the transition matrices (both of the original model and the newly generated piece) with simulated annealing.

In the next subsections, different methods that might be used as quality assessment from a Markov model are described. The first three quality assessment metrics can be used alone. The latter two are constraining metrics that are implemented in combination with one of the first three. These techniques will be implemented and thoroughly evaluated in Section 6.5.

6.3.1 High probability sequences (XE)

Farbood and Schoner (2001) and Lo and Lucas (2006) generate the maximum probability sequence from a statistical model. It makes intuitive sense that this type of sequence is preferred, yet there might be more to a good musical piece than just maximizing the probability (e.g., variety). This will be evaluated in Section 6.5.

Cross-entropy is used as a measure for high probability sequences, whereby minimal cross-entropy corresponds to a maximum likelihood sequence according to the model. The probability \( P(s) \) of a fragment \( s \) consisting of a sequence of notes \( e_1, e_2, \ldots, e_\ell \) is transformed into cross-entropy (Manning and Schutze, 1999). The cross-entropy is the mean of the information content \( h_i \) of each event \( e_i \) in the sequence:

\[
h_i = -\log_2 P(e_i | e_{i-1})
\] (6.4)

\[
f(s) = \frac{1}{\ell - 1} \sum_{i=2}^{\ell} h_i
\] (6.5)

The quality of a musical fragment is thus evaluated according to the cross-entropy (average negative log probability) of the fragment computed using the note transitions of the transition matrix. This forms the objective function
6.3. Using Markov Models within Evaluation Metrics

\( f(s) \) that should be minimized. This is the same approach as the one used in the previous chapter to generate first species counterpoint.

### 6.3.2 Minimal distance between TM of model and solution (DI)

Davismoon and Eccles (2010) use an evaluation metric that tries to match the transition matrices of both the original model and the newly generated piece by minimizing the Euclidean distance between them. This will ensure that they have an equal distribution of probabilities after each possible note. The metric used in this chapter is based on Davismoon and Eccles (2010) and can be formulated as follows for an \( N \times N \) transition matrix:

\[
f(s) = \frac{1}{N} \sqrt{\sum_{a \in \xi} \sum_{b \in \xi} (P(b \mid a) - \bar{P}(b \mid a))^2}
\]

where \( \xi \) is the set of event symbols, for example in the bagana the finger numbers, \( P(b \mid a) \) is the model transition probability from \( a \) to \( b \), and \( \bar{P}(b \mid a) \) is the transition probability calculated from the new piece.

It is expected that this measure enforces more variety in the generated music, as the overall probability transition distribution is optimized to resemble the one of the corpus. The musical output of the VNS that minimizes this metric as its objective function will be evaluated in the experiment in Section 6.5.

### 6.3.3 Delta cross-entropy (DE)

In Subsection 6.3.1 cross-entropy was minimized to find the maximum likelihood sequence. It cannot be guaranteed that this is a sequence a listener would enjoy. If we look at the corpus, there are proportionally fewer pieces with low cross-entropy. Figure 6.3 shows a histogram of the cross-entropy data calculated with leave-one-out cross-validation from the corpus used in the
experiment of Section 6.5. That is, every piece was left out of the corpus, the model retrained, and the cross-entropy of that piece was computed according to the model. It is clear from this figure that most pieces are not even close to the lowest entropy value that occurs in the corpus. As the results in Section 6.5 will indicate, the single minimal cross-entropy sequence can be very repetitive. Optimizing to the average cross-entropy value $E$ might offer a solution for this.

When optimizing towards the average cross-entropy value, the function being minimized thus becomes:

$$f(s) = \left| E - \frac{1}{\ell - 1} \sum_{i=2}^{\ell} h_i \right|$$  \hspace{1cm} (6.7)

where $E$ is the average cross-entropy of the corpus.

Figure 6.3.: Histogram of cross-entropy values of the corpus
6.3. Using Markov Models within Evaluation Metrics

6.3.4 Information contour (i)

One of the problems mentioned by Lo and Lucas (2006) with high probability sequences is that they often sound uninteresting and repetitive. More diversity might be achieved by defining the “information contour” within a piece. Information contour is a measure that describes the movement of information between two successive events (up indicating less expected than the previous event, down indicating more expected than the previous event). It can be seen as the contour of the information flow, which has been used by Witten et al. (1994) and Potter et al. (2007) to measure information dynamics in a musical analysis. In order to measure this a viewpoint is created that expresses if the information content, with respect to a model of the corpus that does not include the template piece for each event, is higher, lower, or equal to that of the previous event. The information contour \( C(e_i) \) of event \( e_i \) is defined as:

\[
C(e_i) = \begin{cases} 
\text{up} & \text{when } h_i > h_{i-1} \\
\text{same} & \text{when } h_i = h_{i-1} \\
\text{down} & \text{when } h_i < h_{i-1}
\end{cases}
\]

In the experiment performed in Section 6.5, the information contour was calculated for each note transition of a selected template song (Tew Semagn Hagere). When evaluating a new solution, a similar information contour may be desirable. Therefore, the objective function to be minimized can be specified as follows for a piece of \( \ell \) notes:

\[
f(s) = M_c \times \sum_{i=2}^{\ell} x_i \] (6.8)

whereby

\[
\begin{align*}
    x_i &= 1 & \text{when } C(e_i) \text{ is not the same as in the template} \\
    x_i &= 0 & \text{when } C(e_i) \text{ is the same as in the template}
\end{align*}
\]
and $M_c$ is an arbitrarily high number.

This metric will be tested in conjunction with the first three metrics by summing the objective functions. By using the arbitrarily high number $M_c$ in the equation above, optimizing the information part will have priority over the other term of the objective function (low entropy, minimize TM distance, delta cross-entropy).

6.3.5 Unwords ($u$)

While music contains patterns that are repeated, it equally contains rare patterns. Conklin (2013a) identified antipatterns, i.e., significantly rare patterns, from a corpus of Basque folk music and from the corpus of bagana music used in this research (Conklin and Weisser, 2014). A related category of rare patterns are those of unwords. Herold et al. (2008), in their paper on genome research, first suggested this term for the shortest words from the underlying alphabet that do not show up in a given sequence. Unwords are thus defined as the shortest sequence of notes (i.e., not contained within a longer unword) that never occur in the corpus. Among these words, we filter for those that are statistically significant. This results in a list of words whose absence from the corpus is surprising given their letter statistics (Conklin and Weisser, 2014). These patterns may represent structural constraints of a style.

A related approach to improve the music generated by simple Markov models is by adding constraints on the subsequences that can be generated. For example, Papadopoulos et al. (2014) efficiently avoid all subsequences greater than a specified maximum order $k$, for the purpose of avoiding simple regeneration of long fragments identical to the corpus. A contrasting approach to this problem is to constrain the types of short words that can be generated based on the analysis of a corpus, i.e., unwords, rather than uniformly forbidding all words of a specified length or greater.

To find unwords, the algorithm of Conklin and Weisser (2014) was used to efficiently search the space of bagana finger patterns for significant unwords. Table 6.1 lists the resulting set of 14 unwords. These unwords, all trigrams,
variable neighbourhood search

Table 6.1.: The set of unwords that were found in the bagana corpus

<table>
<thead>
<tr>
<th>(4, 2, 1)</th>
<th>(2, 1, 2)</th>
<th>(4, 1, 1)</th>
<th>(1, 4, 1)</th>
<th>(3, 4, 1)</th>
<th>(1, 4, 3)</th>
<th>(2, 4, 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 4, 4)</td>
<td>(3, 2, 1)</td>
<td>(2, 1, 1)</td>
<td>(4, 1, 4)</td>
<td>(4, 1, 2)</td>
<td>(1, 2, 5)</td>
<td>(4, 5, 2)</td>
</tr>
</tbody>
</table>

are all formed from one or more bigrams that were identified as antipatterns by Conklin and Weisser (2014). To use these for evaluating music, their occurrence is given a penalty according to the following formula:

\[ f(s) = M_w \times u \]  

(6.9)

whereby \( M_w \) is an arbitrarily high number and \( u \) is the total number of unwords counted in the piece.

This quality measure can be seen more as a hard constraint since unwords never occur in the original corpus. Therefore it is combined with the first three techniques from this section in the experiment. This is done by summing the objective functions for both techniques. The use of an arbitrarily high number \( M_w \) will again give priority to the removal of the unwords over the other metric with which this is combined (low entropy, minimize TM distance, delta cross-entropy).

6.4 VARIABLE NEIGHBOURHOOD SEARCH

When ensuring the long term coherence of a musical piece by imposing a semiotic structure, a simple random walk strategy for generation is no longer an option because only in the infinite limit can it be ensured that random walk will generate a sequence respecting the coherence. Therefore, we turn to an optimization technique in this research, whereby the best possible combination of notes needs to be found to fit a certain style. A bridge between sampling from statistical models and optimizing according to an objective function is made by comparing different quality measures. The resulting problem is a
combinatorial optimization problem which is computationally complex due to the exponential number of possible solutions.

A VNS for generating counterpoint based on formal rules from music theory is developed and implemented by the authors in the first part of this thesis. In the previous chapter, this algorithm has been modified to generate high probability sequences, from which the question arose whether the highest probability sequence is desirable (Herremans et al., 2014b). In this chapter, different evaluation metrics are implemented and the obtained results are discussed.

Figure 6.4.: Overview of the VNS.
The structure of the implemented VNS is represented in Figure 6.4. It is composed of the same components as explained in Section 1.3 and the size of the random perturbation as well as the size of the tabu lists and other parameters were set to the optimum values resulting from a full factorial experiment on first species counterpoint (Herremans and Sörensen, 2012). The VNS algorithm was implemented in C++ and the source code is available online.1

6.5 Results

An experiment was set up in order to compare the outcome of the different evaluation techniques discussed in section 6.3. They were all implemented in the objective function of the VNS described in the previous section. The algorithm stopped after performing 100 000 moves or when no improving solution was found after 1 000 moves.

6.5.1 Training data and Markov model

The corpus used in this experiment is described in more detail by Conklin and Weisser (2014). It consists of 37 pieces of bagana music that have been recorded by Weisser (2005) between 2002 and 2005 in Ethiopia (except for two of them recorded in Washington DC). The songs consist of a relatively short melody, repeated several times with different lyrics, except for the refrain. The entire corpus has been annotated with the repetition structures described in Section 6.2 by a bagana expert.

A piece called Tew Semagn Hagere by Alemu Aga, was selected from the corpus as a template piece. The rhythm within the patterns was kept fixed. The evaluation method based on information contour described in Section 6.3 needs a template to calculate the target information contour. The same piece was also used to get the global structure discussed in Section 6.2.3.

1 http://antor.ua.ac.be/musicvns

The output of the algorithm was rendered in the tezeta scale (Conklin and Weisser, 2014) using F for finger 1 (see Figure 6.1) with a bagana soundfont and presented to Stéphanie Weisser, a bagana expert who evaluated the fragments discussed in Section 6.5.2. Her comments on a preliminary experiment resulted in some improvements of the algorithm, including the fixation of the first note to an A (finger 4) and the last note to a C (finger 2). The results were then presented again for evaluation.

A first order Markov model was learned from the corpus of bagana music. First order models can be weak models, as also stated by Lo and Lucas (2006). Yet in some cases there is not enough data to generate a higher order model, as in the case of the bagana corpus. Working with a first order model allows training on a small corpus, and also gives us a very clear overview of the effects of the different metrics, without having to look at more complicated second order patterns. The resulting transition matrix is represented in Table 6.2.

### Table 6.2: Transition matrix based on the bagana corpus; finger numbers as indices, and corresponding pitch class names (Tezeta scale) in brackets

<table>
<thead>
<tr>
<th></th>
<th>2 (C)</th>
<th>3 (D)</th>
<th>1 (F)</th>
<th>5 (G)</th>
<th>4 (A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 (C)</td>
<td>0.291</td>
<td>0.263</td>
<td>0.015</td>
<td>0.040</td>
<td>0.390</td>
</tr>
<tr>
<td>3 (D)</td>
<td>0.238</td>
<td>0.039</td>
<td>0.694</td>
<td>0.018</td>
<td>0.011</td>
</tr>
<tr>
<td>1 (F)</td>
<td>0.029</td>
<td>0.330</td>
<td>0.237</td>
<td>0.357</td>
<td>0.047</td>
</tr>
<tr>
<td>5 (G)</td>
<td>0.049</td>
<td>0.032</td>
<td>0.401</td>
<td>0.153</td>
<td>0.366</td>
</tr>
<tr>
<td>4 (A)</td>
<td>0.502</td>
<td>0.005</td>
<td>0.005</td>
<td>0.281</td>
<td>0.206</td>
</tr>
</tbody>
</table>

6.5.2 Musical results

The VNS algorithm was run with the different metrics from Section 6.3 as its objective function. The first three metrics were run independently. Then each of these metrics was combined with unwords and information contour. For each metric, the evaluation of cross-entropy and the distance of the transition...
matrices is shown over time (Figure 6.5). The average cross-entropy value $E$ (see Section 6.3.3) of the corpus is also displayed on the plots in this figure as a reference value. The musical output corresponding to each of the runs visualised in Figure 6.5 is displayed in Figures 6.6, 6.7 and 6.8. These music sheets were presented to the bagana expert for evaluation together with the rendered audio files. Table 6.3 shows that the generated music is different from the template piece, where similarity is measured as the percentage of notes that are the same in both the generated piece and the template piece. When measuring the similarity notes at the same position within the given structure were compared.

<table>
<thead>
<tr>
<th></th>
<th>XE</th>
<th>DI</th>
<th>DE</th>
<th>XEu</th>
<th>Dlu</th>
<th>DEu</th>
<th>XEi</th>
<th>Dli</th>
<th>DEi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Similarity (%)</td>
<td>29</td>
<td>36</td>
<td>26</td>
<td>23</td>
<td>29</td>
<td>52</td>
<td>48</td>
<td>36</td>
<td>48</td>
</tr>
<tr>
<td>Cover of range (%)</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Number of unwords</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 6.3.: General characteristics of the generated music displayed in Figures 6.6, 6.7 and 6.8

*High probability sequences (XE)*

Fragment 1 in Figure 6.6 shows the output of minimizing the cross-entropy with the VNS. As also found by Lo and Lucas (2006), the minimal cross-entropy sequence can be very repetitive. According to the transition matrix, the finger transitions corresponding to the note sequences A–C, C–A, F–D and D–F are indeed high probability transitions, still the global result is not the one a listener would enjoy as there is a lot of oscillation. The model generates two high probability transition loops (A–C and D–F). Figure 6.6(a) confirms that minimizing the cross-entropy using VNS causes a rapid decrease in cross-entropy. This is similar to the experiment done by the authors with first species counterpoint in the previous chapter (Herremans et al., 2014b), where it was shown that VNS is an efficient method for generating high probability sequences, and that VNS rapidly converges to the minimum cross-entropy sequence. It is also noticeable from Figure 6.6(a) that optimizing with the
Figure 6.5: Evolution of cross-entropy and distance of transition matrices over time

(a) High probability (XE)  (b) TM distance (DI)  (c) Delta cross-entropy (DE)

(d) High probability with unwords (XEu)  (e) TM distance with unwords (DIu)  (f) Delta cross-entropy with unwords (DEu)

(g) High probability with information contour (XEi)  (h) TM distance with information (DIi)  (i) Delta cross-entropy with information contour (DEi)
Figure 6.6.: Musical output by using the three main evaluation metrics (XE, DI and DE)

Fragment 1: High probability (XE)

Fragment 2: TM distance (DI)

Fragment 3: Delta cross-entropy (DE)

XE metric does not cause a decrease in the DI metric, but rather undirected movement.

**Minimal distance between TM of model and solution (DI)**

When minimizing the distance between the transition matrices of the model and a generated solution with VNS, we again see a rapid decrease in this metric in Figure 6.6(b). The cross entropy measure converts to the average cross-entropy value. This means that by minimizing the DI metric, the cross-entropy moves toward the average value. The music generated music is not too repetitive and the expert listener considered the fragment (Fragment 2 of Figure 6.6) to be very good.
Figure 6.7.: Musical output by using the main three evaluation metrics combined with unwords (u)

Fragment 4: High probability and unwords (XEu)

Fragment 5: TM distance and unwords (Dlu)

Fragment 6: Delta cross-entropy and unwords (DEu)

Delta cross-entropy (DE)

The average of the 37 cross-entropy values, calculated with leave-one-out cross-validation as described in Section 6.3.3, in the bagana corpus is \( E = 1.7 \). The algorithm is able to reach the average cross-entropy value quickly (Figure 6.6(c)). The DI metric is not constrained during DE minimization, and changes randomly throughout the generation process. This is an interesting observation, as minimizing the DI metric in the previous section did constrain both the DI metric to the minimum and the cross-entropy to the average value. This means that optimizing with the DI metric is stronger, more constrained, than solely with the DE metric as it seems to constrain two metrics. The resulting music (Fragment 3 of Figure 6.6) was described by the expert as “not easy to sing with”.
6.5. Results

Figure 6.8.: Musical output by using the main three evaluation metrics combined with information contour (i)

Fragment 7: High probability and information contour (XEi)

Fragment 8: TM distance and information contour (Dli)

Fragment 9: Delta cross-entropy and information contour (DEi)

Unwords (u)

When minimizing the number of unwords together with the three previously discussed metrics, the evolution of the algorithm is very similar (Figures 6.6(d), 6.6(e) and 6.6(f)). This is probably due to the fact that unwords sometimes occur when using the other techniques (see Table 6.3), yet they do not dominate. The high probability sequence still has a lot of repetitions, though slightly decreased.

The expert found the sequence generated with the DEu metric (Fragment 6 of Figure 6.7) very good, with the remark that a player would rather play A–G–F–D in segment $A^3$ instead of A–F–D. This comment is supported by the higher transition probability A–G and G–F versus A–F. The DEu metric optimizes towards the average cross-entropy of the corpus, thus not always preferring the highest probability transitions. The expert also found the segment $A^3$ generated by the DIu metric (Fragment 5 of Figure 6.7) very good. The result with the XEu metric (Fragment 4 of Figure 6.7) is less good as it is too repetitive.
Information contour (i)

Constraining the information contour together with the first three metrics discussed seems to have a positive influence on the quality of the generated music. When minimizing the cross-entropy, it forces the music out of the high probability loops and thus prevents oscillation. This results in a much more varied music (Fragment 7 of Figure 6.8). The plots in Figures 6.6(g), 6.6(h) and 6.6(i) have a similar evolution as before.

The expert found the piece generated with the XEi metric (Fragment 7 of Figure 6.8) very good. The results generated with the Dii metric (Fragment 8 of Figure 6.8) was considered very good, with exception of segment $A^3$ which has some issue with the combination of rhythm and pitch. This is an interesting issue that the authors hope to address in future research by building a statistical model with takes both duration and pitch into account. The piece generated with the DEi metric was considered as good music, with the remark that a player would rather play C–D–F–D instead of C–F–D. Similarly as in the above section, C–D and D–F have much higher transition probabilities than C–F. This can again be explained because the algorithm that was run with the DEi metric (Fragment 9 of Figure 6.8) optimizes towards the average cross-entropy of the corpus instead of the lowest cross-entropy.

6.6 Conclusions

The results of the experiments conducted in this chapter show that there is no single best metric to use in the objective function. Minimizing cross-entropy can lead to oscillating music, a problem which was corrected by combining this metric with information contour. Minimizing the distance between the transition matrix of the model and the generated music also outputs more varied music and seems to constrain the entropy to the average entropy of the corpus. This relationship is not valid in the opposite direction. By constraining the cross-entropy to the average value, the DI metric is not minimized. Optimizing with the DI metric is thus more constraining then optimizing solely with the DE metric. The bagana expert found that
generating with the DI metric produces good musical results. The cross-entropy, TM distance minimization and delta cross-entropy metric all produce good outcomes when combined with information contour. Forbidding the occurrence of unwords in the solution when combined with XE is not enough to avoid oscillations, because in fact even in the corpus extended oscillations do occur, hence they are not significant unwords.

While the comments of the bagana expert are very positive, one possible improvement would be to model and generate into a more complex template with more cyclic patterns. This can equally be handled by the approach used in this chapter simply by specifying an alternative pattern structure for the template piece. It would also be interesting to build a statistical model with takes both note duration and pitch into account. This would address some of the comments of the bagana expert concerning the combination of certain notes with durations.

There are other techniques besides those mentioned above that could be used to improve and measure musical quality of music generated based on a Markov model. One option would be to enforce repetition of note patterns or look at a multiple viewpoint system (Conklin and Witten, 1995; Conklin, 2013b) that includes a viewpoint which models the coherence within finger number sequences. This is already partly implemented on a high level by generating into a certain fixed structure. Another possible idea would be to relax the unwords metric to include antipatterns, i.e., patterns that do occur, but only rarely.

All of the metrics above are based on models created from an entire corpus. Conklin and Witten (1995) additionally consider short term models for which the transition matrix is recalculated based on the newly generated music. This is done for each event, based on the notes before it. This metric might enforce even more diversity as it stimulates repetition and the creation of patterns. This interesting approach is left for future research.

The VNS algorithm allows us to specify a wide variety of constraints. Whenever a neighbourhood is generated, the solutions that do not satisfy these constraints are excluded. This simple mechanism allows the user to implement
many types of constraints, ranging from fixing the pitch of certain notes, to forbidding repetition and only allowing certain pitches.

In this chapter different ways are proposed to construct evaluation metrics based on a Markov model. These metrics are used to evaluate generated bagana music in an optimization procedure. Experiments show that integrating techniques such as information flow, optimizing delta cross-entropy, TM distance minimization and others improve the quality of the generated music based on low order Markov models. A method was also developed that allows the enforcement of a structure and repetition within the music, thus ensuring long term coherence. The methods developed and applied in this paper were applied to music for the Ethiopian bagana and should be applicable to a wide range of musical styles.
Part 3

DANCE HIT PREDICTION
DANCE HIT SONG PREDICTION

Record companies invest billions of dollars in new talent around the globe each year. Gaining insight into what actually makes a hit song would provide tremendous benefits for the music industry. In this research we tackle this question by focusing on the dance hit song classification problem. The focus shifts from music generation to testing to power of machine learning techniques on audio data in this chapter. A database of dance hit songs from 1985 until 2013 is built, including basic musical features, as well as more advanced features that capture a temporal aspect. A number of different classifiers are used to build and test dance hit prediction models. The resulting best model has a good performance when predicting whether a song is a “top 10” dance hit versus a lower listed position.

7.1 Introduction

In 2011 record companies invested a total of 4.5 billion in new talent worldwide (IFPI, 2012). Gaining insight into what actually makes a song a hit would provide tremendous benefits for the music industry. This idea is the main drive behind the new research field referred to as “Hit song science” which Pachet (2012) define as “an emerging field of science that aims at predicting the success of songs before they are released on the market”.

There is a large amount of literature available on song writing techniques (Braheny, 2007; Webb, 1999). Some authors even claim to teach the reader how to write hit songs (Leikin, 2008; Perricone, 2000). Yet very little research has been done on the task of automatic prediction of hit songs or detection of their characteristics.

The increase in the amount of digital music available online combined with the evolution of technology has changed the way in which we listen to music. In order to react to new expectations of listeners who want searchable music collections, automatic playlist suggestions, music recognition systems etc., it is essential to be able to retrieve information from music (Casey et al., 2008).
This has given rise to the field of Music Information Retrieval (MIR), a multidisciplinary domain concerned with retrieving and analysing multifaceted information from large music databases (Downie, 2003).

Many MIR systems have been developed in recent years and applied to a range of different topics such as automatic classification per genre (Tzanetakis and Cook, 2002), cultural origin (Whitman and Smaragdis, 2002), mood (Laurier et al., 2008), composer (Herremans et al., 2013), instrument (Essid et al., 2006), similarity (Schnitzer et al., 2009), etc. An extensive overview is given by Fu et al. (2011). Yet, as it appears, the use of MIR systems for hit prediction remains relatively unexplored.

The first exploration into the domain of hit science is due to Dhanaraj and Logan (2005). They used acoustic and lyric-based features to build support vector machines (SVM) and boosting classifiers to distinguish top 1 hits from other songs in various styles. Although acoustic and lyric data was only available for 91 songs, their results seem promising. The study does however not provide details about data gathering, features, applied methods and tuning procedures.

Based on the claim of the unpredictability of cultural markets made by Salganik et al. (2006), Pachet and Roy (2008) examined the validity of this claim on the music market. Based on a dataset they were not able to develop an accurate classification model for low, medium or high popularity based on acoustic and human features. They suggest that the acoustic features they used are not informative enough to be used for aesthetic judgements and suspect that the previously mentioned study (Dhanaraj and Logan, 2005) is based on spurious data or biased experiments.

Borg and Hokkanen (2011) draw similar conclusions as Pachet and Roy (2008). They tried to predict the popularity of music videos based on their YouTube view count by training support vector machines but were not successful.

Another experiment was set up by Ni et al. (2011), who claim to have proven that hit song science is once again a science. They were able to obtain more optimistic results by predicting if a song would reach a top 5 position on the UK top 40 singles chart compared to a top 30-40 position. The shifting
perceptron model that they built was based on thus far novel audio features mostly extracted from The Echo Nest\textsuperscript{1}. Though they describe the features they used on their website (Jehan and DesRoches, 2012), the paper is very short and does not disclose a lot of details about the research such as data gathering, preprocessing, detailed description of the technique used or its implementation.

In this research accurate models are built to predict if a song is a top 10 dance hit or not. For this purpose, a dataset of dance hits including some unique audio features is compiled. Based on this data different efficient models are built and compared. To the authors’ knowledge, no previous research has been done on the dance hit prediction problem.

In the next section, the dataset used in this chapter is elaborately discussed. In Section 7.3 the data is visualized in order to detect some temporal patterns. Finally, the experimental setup is described and a number of models are built and tested.

7.2 Datasets

The dataset used in this research was gathered in a few stages. The first stage involved determining which songs can be considered as hit songs versus which songs cannot. Secondly, detailed information about musical features was obtained for both aforementioned categories.

7.2.1 Hit listings

Two hit archives available online were used to create a database of dance hits (see Table 7.1). The first one is the singles dance archive from the Official Charts Company (OCC)\textsuperscript{2}. The Official Charts Company is operated by both

\footnotesize{1 echonest.com
2 officialcharts.com}
Table 7.1.: Hit listings overview

<table>
<thead>
<tr>
<th></th>
<th>OCC</th>
<th>BB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top</td>
<td>40</td>
<td>10</td>
</tr>
<tr>
<td>Hit listings</td>
<td>7,159</td>
<td>14,533</td>
</tr>
<tr>
<td>Unique songs</td>
<td>759</td>
<td>3,361</td>
</tr>
</tbody>
</table>

Table 7.2.: Example of hit listings before adding musical features

<table>
<thead>
<tr>
<th>Song title</th>
<th>Artist</th>
<th>Position</th>
<th>Date</th>
<th>Peak position</th>
</tr>
</thead>
<tbody>
<tr>
<td>Harlem Shake</td>
<td>Bauer</td>
<td>2</td>
<td>09/03/13</td>
<td>1</td>
</tr>
<tr>
<td>Are You Ready For Love</td>
<td>Elton John</td>
<td>40</td>
<td>08/12/12</td>
<td>34</td>
</tr>
<tr>
<td>The Game Has Changed</td>
<td>Daft Punk</td>
<td>32</td>
<td>18/12/10</td>
<td>32</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

the British Phonographic Industry and the Entertainment Retailers Association ERA. Their charts are produced based on sales data from retailers through market researcher Millward Brown. The second source is the singles dance archive from Billboard (BB). Billboard is one of the oldest magazines in the world devoted to music and the music industry.

The information was parsed from both websites using the Open source Java html parser library JSoup (Houston, 2013) and resulted in a dataset of 21,692 (7,159 + 14,533) listings with 4 features: song title, artist, position and date. A very small number of hit listings could not be parsed and these were left out of the dataset. The peak chart position for each song was computed and added to the dataset as a fifth feature. Table 7.2 shows an example of the dataset at this point.

3 billboard.com
7.2.2 Feature extraction and calculation

The Echo Nest\footnote{echonest.com} was used in order to obtain musical characteristics for the song titles obtained in previous subsection. The Echo Nest is the world’s leading music intelligence company and has over a trillion data points on over 34 million songs in its database. Its services are used by industry leaders such as Spotify, Nokia, Twitter, MTV, EMI and more (EchoNest, 2013). Bertin-Mahieux et al. (2011) used The Echo Nest to build The One Million Song dataset, a very large freely available dataset that offers a collection of audio features and meta-information for a million contemporary popular songs.

In this research The Echo Nest was used to build a new database mapped to the hit listings. The Open Source java client library jEN for the Echo Nest developer API was used to query the songs (Lamere, 2013). Based on the song title and artist name, The Echo Nest database and Analyzer were queried for each of the parsed hit songs. After some manual and java-based corrections for spelling irregularities (e.g., Featuring, Feat, Ft.) data was retrieved for 697 out of 759 unique songs from the OCC hit listings and 2,755 out of 3,361 unique songs from the BB hit listings. The songs with missing data were removed from the dataset. The extracted features can be divided into three categories: meta-information, basic features from The Echo Nest Analyzer and temporal features.

Meta-information

The first category is meta-information such as artist location, artist familiarity, artist hotness, song hotness etc. This is descriptive information about the song, often not related to the audio signal itself. One could follow the statement of IBM’s Bob Mercer in 1985 “There is no data like more data” (Jelinek, 2005). Yet, for this research, the meta-information is discarded when building the classification models. In this way, the model can work with unknown songs, based purely on audio signals.
Basic analyzer features

The next category consists of basic features extracted by The Echo Nest Analyzer (Jehan and DesRoches, 2012). Most of these features are self-explanatory, except for energy and danceability, of which The Echo Nest did not yet release the formula.

**duration** Length of the track in seconds.

**tempo** The average tempo expressed in beats per minute (bpm).

**time signature** A symbolic representation of how many beats there are in each bar.

**mode** Describes if a song’s modality is major (1) or minor (0).

**key** The estimated key of the track, represented as an integer.

**loudness** The loudness of a track in decibels (dB), which correlates to the psychological perception of strength (amplitude).

**danceability** Calculated by The Echo Nest, based on beat strength, tempo stability, overall tempo, and more.

**energy** Calculated by The Echo Nest, based on loudness and segment durations.

A more detailed description of these Echo Nest features is given by Jehan and DesRoches (2012).
Temporal features

A third category of features was added to incorporate the temporal aspect of the following basic features offered by the Analyzer:

**Timbre** A 12-dimensional vector which captures the tone colour for each segment of a song. A segment is a sound entity (typically under a second) relatively uniform in timbre and harmony.

**Beatdiff** The time difference between subsequent beats.

Timbre is a very perceptual feature that is sometimes referred to as tone colour. In The Echo Nest, 13 basis vectors are available that are derived from the principal components analysis (PCA) of the auditory spectrogram (Jehan, 2005). The first vector of the PCA is referred to as loudness, as it is related to the amplitude. The following 12 basis vectors are referred to as the timbre vectors. The first one can be interpreted as brightness, as it emphasizes the ratio of high frequencies versus low frequencies, a measure typically correlated to the “perceptual” quality of brightness. The second timbre vector has to do with flatness and narrowness of sound (attenuation of lowest and highest frequencies). The next vector represents the emphasis of the attack (sharpness) (EchoNest, 2013). The timbre vectors after that are harder to label, but can be understood by the spectral diagrams given by Jehan (2005).

In order to capture the temporal aspect of timbre throughout a song Schindler and Rauber (2012) introduce a set of derived features. They show that genre classification can be significantly improved by incorporating the statistical moments of the 12 segment timbre descriptors offered by The Echo Nest. In this research the statistical moments were calculated together with some extra descriptive statistics: mean, variance, skewness, kurtosis, standard deviation, 80th percentile, min, max, range and median.

Ni et al. (2013) introduce a variable called Beat CV in their model, which refers to the variation of the time between the beats in a song. In this research, the temporal aspect of the time between beats (beatdiff) is taken into account in
7.3 Evolution over time

A more complete way, using all the descriptive statistics from the previous paragraph.

After discarding the meta-information, the resulting dataset contained 139 usable features. In the next section, these features were analysed to discover their evolution over time.

7.3 Evolution over time

The dominant music that people listen to in a certain culture changes over time. It is no surprise that a hit song from the 60s will not necessarily fit in the contemporary charts. Even if we limit ourselves to one particular style of hit songs, namely dance music, a strong evolution can be distinguished between popular 90s dance songs and this week’s hit. In order to verify this statement and gain insight into how characteristics of dance music have changed, the Billboard dataset (BB) with top 10 dance hits from 1985 until now was analysed.

A dynamic chart was used to represent the evolution of four features over time (Google, 2013). Figure 7.1 shows a screenshot of the Google motion chart5 that was used to visualize the time series data. This graph integrates data mining and information visualization in one discovery tool as it reveals interesting patterns and allows the user to control the visual presentation, thus following the recommendation made by Shneiderman (2002). The x-axis shows the duration and the y-axis is the average loudness per year in Figure 7.1. Additional dimensions are represented by the size of the bubbles (brightness) and the colour of the bubbles (tempo).

Since a motion chart is a dynamic tool that should be viewed on a computer, a selection of features were extracted to more traditional 2-dimensional graphs with linear regressions (see Figure 7.2). Since the OCC dataset contains 3,361 unique songs, the selected features from these songs were averaged per year in order to limit the amount of data points on the graph. A rising trend can

---

5 Interactive motion chart available at http://antor.ua.ac.be/dance
be detected for the loudness, tempo and 1st aspect of timbre (brightness). The correlation between loudness and tempo is in line with the rule proposed by Todd (1992) “The faster the louder, the softer the slower”. Not all features have an apparent relationship with time. Energy, for instance, (see Figure 7.2(e)), doesn’t seem to be correlated with time. It is also remarkable that the danceability feature computed by The Echo Nest decreases over time for dance hits. Since no detailed formula was given by The Echo Nest for danceability, this trend cannot be explained.

Figure 7.2(b) shows that the average tempo in beats per minute (bpm) per year is between 114 and 124 bpm. While Fraisse (1982) found a preference for tempi around 100 bpm, a more recent study by Moelants (2003) found it to be around 120 bpm. For dance music however, the tempo preference is even
higher, around 130 bpm. In Figure 7.2(b) the average tempo of dance hit songs per year is slightly under these values. In order to examine this in more detail, the frequency distribution of the bpm for all songs is displayed in Figure 7.3. This graph shows a noticeable gap between 120–130 and 80–90 bpm. This gap can indicate that there are a few songs with a slow tempo in the dataset, or The Echo Nest Analyzer has mistakenly detected the half-tempo for a few songs. The first is certainly true for a few songs, such as Bedtime Story by Madonna (112 bpm). The latter seems to be true as well for some instances, such as Dog Days Are Over by Florence The Machine (75 bpm). Half-tempo detection is a common mistake in beat detection algorithms whereby a tempo half as slow as the real tempo is detected (Goto and Muraoka, 1997). While the actual values might be a little bit higher in Figure 7.2(b), the rising trend will remain valid, provided that the number of half-tempo errors is similar per year. Later on when classification models are built, care should be taken when interpreting features related to beats per minute. Since the tempo of only a very small number of songs is halved and we can expect the Analyzer to treat similar songs with the same error ratio, the influence on the classification models will be minimal.

The next sections describes an experiment which compares several hit prediction models built in this research.

7.4 DANCE HIT PREDICTION

In this section the experimental setup and preprocessing techniques are described for the classification models built in Section 7.5.

7.4.1 Experiment setup

Figure 7.4 shows a graphical representation of the setup of the experiment described in Section 7.6.1. The dataset used for the hit prediction models in this section is based on the OCC listings. The reason for this is that this
Figure 7.2.: Evolution over time of selected characteristics of top 10 songs
data contains top 40 songs, not just top 10. This will allow us to create a “gap” between the two classes. Since the previous section showed that the characteristics of hit songs evolve over time it is not representable to use data from 1985 for predicting contemporary hits. The dataset used for building the prediction models consists of dance hit songs from 2009 until 2013.

The peak chart position of each song was used to determine if they are a dance hit or not. Three datasets were made with each a different gap between the two classes (see Table 7.3). In the first dataset (D1), hits are considered to be songs with a peak position in the top 10. Non-hits are those that only reached a position between 30 and 40. In the second dataset (D2), the gap between hits and non-hits is smaller, as songs reaching a top position of 20 are still considered to be non-hits. Finally, the original dataset is split in two at position 20, without a gap to form the third dataset (D3). The reason for not

Figure 7.3.: Frequency distribution of the beats per minute feature over the OCC dataset
comparing a top 10 hit with a song that did not appear in the charts is to avoid doing accidental genre classification. If a hit dance song would be compared to a song that does not occur in the hit listings, a second classification model would be needed to ensure that this non-hit song is in fact a dance song. If not, the developed model might distinguish songs based on whether or not they are a dance song instead of a hit. However, it should be noted that not all songs on the dance hit lists are in fact the same type of dance songs, there might be subgenres. Still, they will probably share more common attributes than songs from a random style, thus reducing the noise in the hit classification model. The sizes of the three datasets are listed in Table 7.3, the difference in size can be explained by the fact that songs are excluded in D1 and D2 to form the gap. In the next sections, models are built and compare the performance of classifiers on these three datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Hits</th>
<th>Non-hits</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>Top 10</td>
<td>Top 30-40</td>
<td>400</td>
</tr>
<tr>
<td>D2</td>
<td>Top 10</td>
<td>Top 20-40</td>
<td>550</td>
</tr>
<tr>
<td>D3</td>
<td>Top 20</td>
<td>Top 20-40</td>
<td>697</td>
</tr>
</tbody>
</table>

The Open Source software Weka was used to create the models (Witten and Frank, 2005). Weka’s toolbox and framework is recognized as a landmark system in the data mining and machine learning field (Hall et al., 2009).

7.4.2 Preprocessing

The class distribution of the three datasets used in the experiment is displayed in Figure 7.5. Although the distribution is not heavily skewed, it is not completely balanced either. Because of this the use of the accuracy measure to evaluate our results is not suited and the area under the receiver operating curve (AUC) (Fawcett, 2004) was used instead (see section 7.6).
All of the features in the datasets were standardized using statistical normalization and feature selection was done (see Figure 7.4), using the procedure CfsSubsetEval from Weka with GeneticSearch. This procedure uses the individual predictive ability of each feature and the degree of redundancy between them to evaluate the worth of a subset of features (Hall, 1999). Feature selection was done in order to avoid the “curse of dimensionality” by having a very sparse feature set. McKay and Fujinaga (2006) point to the fact that having a limited amount of features allows for a thorough testing of the model with limited instances and can thus improve the quality of the classification model. Added benefits are the improved comprehensibility of a model with a limited
amount of highly predictive variables (Hall, 1999) and better performance of the learning algorithm (Piramuthu, 2004).

The feature selection procedure in Weka reduces the data to 35–50 attributes, depending on the dataset. The most commonly occurring features after feature selection are listed in Table 7.4. Interesting to note is that the features danceability and energy both disappear from the reduced datasets, except for danceability which stays in the D3 dataset. This could be explained by the fact that these features are calculated by The Echo Nest based on other features.

7.5 Classification Techniques

A total of five models were built for each dataset using diverse classification techniques. The two first models (decision tree and ruleset) can be considered as the easiest to understand classification models due to their linguistic nature (Martens, 2008). The other three models focus on accurate prediction. In the following subsections, the individual algorithms are briefly discussed.
Table 7.4.: The most commonly occurring features in D1, D2 and D3 after FS

<table>
<thead>
<tr>
<th>Feature</th>
<th>Occurance</th>
<th>Feature</th>
<th>Occurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beatdiff (range)</td>
<td>3</td>
<td>Timbre 1 (mean)</td>
<td>2</td>
</tr>
<tr>
<td>Timbre 1 (80 perc)</td>
<td>3</td>
<td>Timbre 1 (median)</td>
<td>2</td>
</tr>
<tr>
<td>Timbre 1 (max)</td>
<td>3</td>
<td>Timbre 2 (max)</td>
<td>2</td>
</tr>
<tr>
<td>Timbre 1 (std dev)</td>
<td>3</td>
<td>Timbre 2 (mean)</td>
<td>2</td>
</tr>
<tr>
<td>Timbre 2 (80 perc)</td>
<td>3</td>
<td>Timbre 2 (range)</td>
<td>2</td>
</tr>
<tr>
<td>Timbre 3 (mean)</td>
<td>3</td>
<td>Timbre 3 (var)</td>
<td>2</td>
</tr>
<tr>
<td>Timbre 3 (median)</td>
<td>3</td>
<td>Timbre 4 (80 perc)</td>
<td>2</td>
</tr>
<tr>
<td>Timbre 3 (min)</td>
<td>3</td>
<td>Timbre 5 (mean)</td>
<td>2</td>
</tr>
<tr>
<td>Timbre 3 (std dev)</td>
<td>3</td>
<td>Timbre 5 (std dev)</td>
<td>2</td>
</tr>
<tr>
<td>Beatdiff (80 perc)</td>
<td>2</td>
<td>Timbre 6 (median)</td>
<td>2</td>
</tr>
<tr>
<td>Beatdiff (std dev)</td>
<td>2</td>
<td>Timbre 6 (range)</td>
<td>2</td>
</tr>
<tr>
<td>Beatdiff (var)</td>
<td>2</td>
<td>Timbre 6 (var)</td>
<td>2</td>
</tr>
<tr>
<td>Timbre 11 (80 perc)</td>
<td>2</td>
<td>Timbre 7 (var)</td>
<td>2</td>
</tr>
<tr>
<td>Timbre 11 (var)</td>
<td>2</td>
<td>Timbre 8 (Median)</td>
<td>2</td>
</tr>
<tr>
<td>Timbre 12 (kurtosis)</td>
<td>2</td>
<td>Timbre 9 (kurtosis)</td>
<td>2</td>
</tr>
<tr>
<td>Timbre 12 (Median)</td>
<td>2</td>
<td>Timbre 9 (max)</td>
<td>2</td>
</tr>
<tr>
<td>Timbre 12 (min)</td>
<td>2</td>
<td>Timbre 9 (Median)</td>
<td>2</td>
</tr>
</tbody>
</table>
together with their main parameters and settings, followed by a comparison in Section 7.6. The AUC values mentioned in this section are based on 10-fold cross-validation performance (Witten and Frank, 2005). The shown models are built on the entire dataset.

7.5.1 C4.5 tree

A decision tree for dance hit prediction was built with J48, Weka’s implementation of the popular C4.5 algorithm (Witten and Frank, 2005).

A “divide and conquer” approach is used by the C4.5 algorithm to build trees recursively (Quinlan, 1993). This is a top down approach, in which a feature is sought that best separates the classes, followed by pruning of the tree (Wu et al., 2008). This pruning is performed by a subtree raising operation in an inner cross-validation loop (3 folds by default in Weka) (Witten and Frank, 2005).

For the comparative tests in Section 7.6 Weka’s default settings were kept for J48. In order to create a simple abstracted model on dataset D1 (FS) for visual insight in the important features, a less accurate model (AUC 0.54) was created by pruning the tree to depth four. The resulting tree is displayed in Figure 7.6. It is noticeable that time differences between the third, fourth and ninth timbre vector seem to be important features for classification.

7.5.2 RIPPER ruleset

Much like trees, rulesets are a useful tool to gain insight in the data. In this section JRip, Weka’s implementation of the propositional rule learner RIPPER (Cohen, 1995), was used to inductively build “if-then” rules. The “Repeated Incremental Pruning to Produce Error Reduction algorithm” (RIPPER), uses sequential covering to generate the ruleset. In a first step of this
algorithm, one rule is learned and the training instances that are covered by this rule are removed. This process is then repeated (Hall et al., 2009).

The ruleset displayed in Table 7.5 was generated with Weka’s default parameters for number of data instances (2) and folds (3) (AUC = 0.56 on dataset D1, see Table 7.7). It’s notable that the third timbre vector is an important feature
again. It would appear that this feature should not be underestimated when composing dance songs.

7.5.3 Naive Bayes

The naive Bayes classifier estimates the probability of a hit or non-hit based on the assumption that the features are conditionally independent. This conditional independence assumption is represented by equation (7.1) given class label $y$ (Tan et al., 2007).

$$P(x|Y = y) = \prod_{j=1}^{M} P(x_j|Y = y), \quad (7.1)$$

whereby each attribute set $x = \{x_1, x_2, \ldots, x_N\}$ consists of $M$ attributes.

Because of the conditional dependence assumption, the class-conditional probability for every combination of $X$ does not need to be calculated. Only the conditional probability of each $x_i$ given $Y$ has to be estimated. This offers a practical advantage since a good estimate of the probability can be obtained without the need for a very large training set.

Naive Bayes classifies a test record by calculating the posterior probability for each class $Y$ (Lewis, 1998):

$$P(Y|x) = \frac{P(Y) \cdot \prod_{j=1}^{M} P(x_j|Y)}{P(x)} \quad (7.2)$$

Although this independence assumption is generally a poor assumption in practice, numerous studies prove that naive Bayes competes well with more sophisticated classifiers (Rish, 2001). In particular, naive Bayes seems to be particularly resistant to isolated noise points, robust to irrelevant attributes, but its performance can degrade by correlated attributes (Tan et al., 2007).
Table 7.7 confirms that Naive Bayes performs very well, with an AUC of 0.65 on dataset D1 (FS).

7.5.4 Logistic regression

The SimpleLogistic function in Weka was used to build a logistic regression model (Witten and Frank, 2005).

Equation (7.3) shows the output of a logistic regression, whereby $f_{hit}(s_i)$ represents the probability that a song $i$ with $M$ features $x_j$ is a dance hit. This probability follows a logistic curve, as can be seen in Figure 7.7. The cut-off point of 0.5 will determine if a song is classified as a hit or a non-hit. With AUC = 0.65 for dataset D1 and AUC=0.67 for dataset D2 (see Table 7.7), logistic regression performs best for this particular classification problem.

\[
f_{hit}(s_i) = \frac{1}{1 + e^{-s_i}} \quad \text{whereby} \quad s_i = b + \sum_{j=1}^{M} a_j \cdot x_j \tag{7.3}
\]

![Figure 7.7.: Probability that song $i$ is a dance hit](image)

Logistic regression models generally require limited computing power and are less prone to overfitting than other models such as neural networks (Tu, 1996).
7.5.5 Support vector machines

Weka’s sequential minimal optimization algorithm (SMO) was used to build two support vector machine classifiers. The support vector machine (SVM) is a learning procedure based on the statistical learning theory (Vapnik, 1995). Given a training set of \( N \) data points \( \{ (x_i, y_i) \}_{i=1}^{N} \) with input data \( x_i \in \mathbb{R}^n \) and corresponding binary class labels \( y_i \in \{ -1, +1 \} \), the SVM classifier should fulfill following conditions (Cristianini and Shawe-Taylor, 2000; Vapnik, 1995):

\[
\begin{align*}
 w^T \varphi(x_i) + b &\geq +1, & \text{if } y_i = +1 \\
 w^T \varphi(x_i) + b &\leq -1, & \text{if } y_i = -1
\end{align*}
\] (7.4)

which is equivalent to

\[
y_i [w^T \varphi(x_i) + b] \geq 1, \quad i = 1, \ldots, N. \tag{7.5}
\]

The non-linear function \( \varphi(\cdot) \) maps the input space to a high (possibly infinite) dimensional feature space. In this feature space, the above inequalities basically construct a hyperplane \( w^T \varphi(x) + b = 0 \) discriminating between the two classes. By minimizing \( w^T w \), the margin between both classes is maximized.

In primal weight space the classifier then takes the form

\[
y(x) = \text{sign}[w^T \varphi(x) + b], \quad (7.6)
\]

but, on the other hand, is never evaluated in this form. One defines the convex optimization problem:

\[
\min_{w,b,\xi} J(w, b, \xi) = \frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i \tag{7.7}
\]

subject to

\[
\begin{align*}
 y_i [w^T \varphi(x_i) + b] &\geq 1 - \xi_i, & i = 1, \ldots, N \\
 \xi_i &\geq 0, & i = 1, \ldots, N.
\end{align*}
\] (7.8)
The variables $\xi_i$ are slack variables which are needed to allow misclassifications in the set of inequalities (e.g., due to overlapping distributions). The first part of the objective function tries to maximize the margin between both classes in the feature space and is a regularisation mechanism that penalizes for large weights, whereas the second part minimizes the misclassification error. The positive real constant $C$ is the regularisation coefficient and should be considered as a tuning parameter in the algorithm.

This leads to the following classifier (Cristianini and Shawe-Taylor, 2000):

$$y(x) = \text{sign}\left[ \sum_{i=1}^{N} \alpha_i y_i K(x_i, x) + b \right],$$  

(7.9)

whereby $K(x_i, x) = \varphi(x_i)^T \varphi(x)$ is taken with a positive definite kernel satisfying the Mercer theorem. The Lagrange multipliers $\alpha_i$ are then determined by
optimalizing the dual dual problem. The following kernel functions $K(\cdot, \cdot)$ were used:

\[
K(x, x_i) = (1 + x_i^T x / c)^d, \quad \text{(polynomial kernel)}
\]

\[
K(x, x_i) = \exp\{-\|x - x_i\|^2 / \sigma^2\}, \quad \text{(RBF kernel)}
\]

where $d, c$ and $\sigma$ are constants.

For low-noise problems, many of the $\alpha_i$ will typically be equal to zero (sparseness property). The training observations corresponding to non-zero $\alpha_i$ are called support vectors and are located close to the decision boundary.

As equation (7.9) shows, the SVM classifier with non-linear kernel is a complex, non-linear function. Trying to comprehend the logics of the classifications made is quite difficult, if not impossible (Martens et al., 2009; Martens and Provost, 2014).

In this research, the polynomial kernel and RBF kernel were used to build the models. Although Weka’s default settings were used in the previous models, the hyperparameters for the SVM model were optimized. To determine the optimal settings for the regularisation parameter $C (1, 3, 5, \ldots 21)$, the $\sigma$ for the RBF kernel ($\frac{1}{\sigma^2} = 0.00001, 0.0001, \ldots 10$) and the exponent $d$ for the polynomial kernel (1, 2), GridSearch was used in Weka. The choice of hyperparameters to test was inspired by settings suggesting by Weka (2013b). GridSearch performs 2-fold cross-validation on the initial grid. This grid is determined by the two input parameters ($C$ and $\sigma$ for the RBF kernel, $C$ and $d$ for the polynomial kernel). 10-fold cross-validation is then performed on the best point of the grid based on the weighted AUC by class size and its adjacent points. If a better pair is found, the procedure is repeated on its neighbours until no better pair is found or the border of the grid is reached (Weka, 2013a). This hyperparameter optimization is performed in the “classification model” box in Figure 7.4. The resulting AUC-value is 0.59 for the SVM with polynomial and 0.56 for the SVM with RBF kernel on D1 (FS) (see Table 7.7).
In this section, two experiments are described. The first one builds models for all of the datasets (D1, D2 & D3), both with and without feature selection. The evaluation is done by taking the average of 10 runs, each with a 10-fold cross-validation procedure. In the second experiment, the performance of the classifiers on the best dataset is compared with an out-of-time test set.

7.6.1 Full experiment with cross-validation

A comparison of the accuracy and the AUC is displayed in Table 7.6 and 7.7 for all of the above mentioned classifiers. The tests were run 10 times, each time with stratified 10-fold cross-validation (10CV), both with and without feature selection (FS). This process is depicted in Figure 7.4. As mentioned in Section 7.4.2, AUC is a more suited measure since the datasets are not entirely balanced (Fawcett, 2004), yet both are displayed to be complete. During the cross-validation procedure, the dataset is divided into 10 folds. 9 of them are used for model building and 1 for testing. This procedure is repeated 10 times. The displayed AUC and accuracy in this subsection are the average results over the 10 test sets and the 10 runs. The resulting model is built on the entire dataset and can be expected to have a performance which is at least as good as the 10CV performance. A total of 10 runs were performed with the 10CV procedure and the average results are displayed in Table 7.6 and 7.7. A Wilcoxon signed-rank test is conducted to compare the performance of the models with the best performing model. The null hypothesis of this test states: “There is no difference in the performance of a model with the best model”.

As described in the previous section, decision trees and rulesets do not always offer the most accurate classification results, but their main advantage is their comprehensibility (Craven and Shavlik, 1996). It is rather surprising that support vector machines do not perform very well on this particular problem. The overall best technique seems to be the logistic regression, closely followed by naive Bayes. Another conclusion from the table is that feature selection
Table 7.6.: Results with 10-fold validation (accuracy)

<table>
<thead>
<tr>
<th>Accuracy (%)</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>FS</td>
<td>-</td>
</tr>
<tr>
<td>C4.5</td>
<td>57.05</td>
<td>58.25</td>
<td>54.95</td>
</tr>
<tr>
<td>RIPPER</td>
<td>60.95</td>
<td>62.43</td>
<td>56.69</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>65</td>
<td>65</td>
<td>60.22</td>
</tr>
<tr>
<td>Logistic regression</td>
<td>64.65</td>
<td>64</td>
<td>62.64</td>
</tr>
<tr>
<td>SVM (Polynomial)</td>
<td>64.97</td>
<td>64.7</td>
<td>61.55</td>
</tr>
<tr>
<td>SVM (RBF)</td>
<td>64.7</td>
<td>64.63</td>
<td>59.8</td>
</tr>
</tbody>
</table>

FS = feature selection, $p < 0.01$: italic, $p > 0.05$: bold, best: bold.

Table 7.7.: Results for 10 runs with 10-fold validation (AUC)

<table>
<thead>
<tr>
<th>AUC</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>FS</td>
<td>-</td>
</tr>
<tr>
<td>C4.5</td>
<td>0.53</td>
<td>0.55</td>
<td>0.55</td>
</tr>
<tr>
<td>RIPPER</td>
<td>0.55</td>
<td>0.56</td>
<td>0.56</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>0.64</td>
<td>0.65</td>
<td>0.64</td>
</tr>
<tr>
<td>Logistic regression</td>
<td>0.65</td>
<td>0.65</td>
<td>0.67</td>
</tr>
<tr>
<td>SVM (Polynomial)</td>
<td>0.6</td>
<td>0.59</td>
<td>0.61</td>
</tr>
<tr>
<td>SVM (RBF)</td>
<td>0.56</td>
<td>0.56</td>
<td>0.59</td>
</tr>
</tbody>
</table>

FS = feature selection, $p < 0.01$: italic, $p > 0.05$: bold, best: bold.
7.6. RESULTS

seems to have a positive influence on the AUC for D1 and D3. As expected, the overall best results when taking into account both AUC and accuracy can be obtained using the dataset with the biggest gap, namely D1.

Table 7.8: Results for 10 runs on D1 (FS) with 10-fold cross-validation compared with the split test set

<table>
<thead>
<tr>
<th></th>
<th>AUC</th>
<th>accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>split 10CV</td>
<td>split 10CV</td>
</tr>
<tr>
<td>C4.5</td>
<td>0.62 0.55</td>
<td>62.50 58.25</td>
</tr>
<tr>
<td>RIPPER</td>
<td>0.66 0.56</td>
<td>85 62.43</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>0.79 0.65</td>
<td>77.50 65</td>
</tr>
<tr>
<td>Logistic regression</td>
<td>0.81 0.65</td>
<td>80 64</td>
</tr>
<tr>
<td>SVM (Polynomial)</td>
<td>0.729 0.59</td>
<td>85 64.7</td>
</tr>
<tr>
<td>SVM (RBF)</td>
<td>0.57 0.56</td>
<td>8.2 64.63</td>
</tr>
</tbody>
</table>

$p < 0.01$: italic, $p > 0.05$: bold, best: **bold**.

The overall best model seems to be logistic regression. The receiver operating curve (ROC) is displayed in Figure 7.9. The ROC curve displays the trade-off between true positive rate (TPR) and false negative rate (FNR) of the logistic classifier with 10-fold cross-validation for D1 (FS). The model clearly scores better than a random classification, which is represented by the diagonal through the origin.

The confusion matrix of the logistic regression shows that 209 hits (i.e. 83% of the actual hits) were accurately classified as hits and 47 non-hits classified as non-hits (i.e. 32% of the actual non-hits). Yet overall, the model is able to make a fairly good distinction between classes, which proves that the dance hit prediction problem can be tackled as realistic top 10 versus top 30-40 classification problem with logistic regression.
CHAPTER 7. DANCE HIT SONG PREDICTION

Figure 7.9: ROC for logistic regression

Figure 7.10: Class distribution of the split training and test sets


### 7.6.2 Experiment with out-of-time test set

A second experiment was conducted with an out-of-time test set based on D1 with feature selection. The instances were first ordered by date, and then split into a 90% training and 10% test set. Table 7.8 confirms the good performance of the logistic regression. A peculiar observation from this table is that the model seems to be able to predict better for newer songs (AUC: 0.81 versus 0.65). This can be due to coincidence, different class distribution between training and test set (see Figure 7.10) or the structure of the dataset. One speculation of the authors is that the oldest instances of the dataset might be “lingering” hits, meaning that they were top 10 hits on a date before the earliest entry in the dataset, and were still present in a low position in the used hit listings. These songs would be falsely seen as non-hits, which might cause the model to predict less good for older songs. Some results for more recent dance hits, not included in the training or test set, are displayed in Table 7.10. It must be noted that since these are very recent songs, they might still go up in the charts.

### 7.7 Conclusions

Multiple models were built that can successfully predict if a dance song is going to be a top 10 hit versus a lower positioned dance song. In order to do this, hit listings from two chart magazines were collected and mapped to audio features provided by The Echo Nest. Standard audio features were used, as well as more advanced features that capture the temporal aspect. This resulted in a model that could accurately predict top 10 dance hits.
Table 7.10.: Probability of recent dance songs being a top 10 hit according to the logistic regression model (D1)

<table>
<thead>
<tr>
<th>Title</th>
<th>Artist</th>
<th>Top position</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gangnam Style</td>
<td>Psy</td>
<td>1</td>
<td>0.79</td>
</tr>
<tr>
<td>Bailando</td>
<td>Enrique Iglesias Ft Descemer Bueno &amp; Gente de Zona</td>
<td>1</td>
<td>0.84</td>
</tr>
<tr>
<td>Rather Be</td>
<td>Clean Bandit Featuring Jess Glynne</td>
<td>2</td>
<td>0.78</td>
</tr>
<tr>
<td>Get Lucky</td>
<td>Daft Punk</td>
<td>1</td>
<td>0.61</td>
</tr>
<tr>
<td>Rock This Party</td>
<td>Fuzztroniks</td>
<td>12</td>
<td>0.85</td>
</tr>
<tr>
<td>Outside</td>
<td>Calvin Harris Ft Ellie Goulding</td>
<td>15</td>
<td>0.84</td>
</tr>
</tbody>
</table>

The top hit position is based on data from Billboard until November 5th 2014.

This research proves that popularity of dance songs can be learnt from the analysis of music signals. Previous less successful results in this field speculate that their results could be due to features that are not informative enough (Pachet and Roy, 2008). The positive results from this research could indeed be due to the use of more advanced temporal features. A second cause might be the use of “recent” songs only, which eliminates the fact that hit music evolves over time. It might also be due to the nature of dance music or that by focusing on one particular style of music, any noise created by classifying hits of different genres is reduced. Finally, by comparing different classifiers that have significantly different results in performance, the best model could be selected.

This model was implemented in an online application where users can upload their audio data and get the probability of it being a hit. An interesting future expansion would be to improve the accuracy of the model by including more features such as lyrics, social network information and others. The model could also be expanded to predict hits of other musical styles. In the line of research being done with automatic composition systems (see previous chapters), it is also interesting to see if the classification models from this chapter could be included in an optimization function (e.g., a type of fitness function) and used to generate new dance hits or improve existing ones.

http://antor.ua.ac.be/dance
CONCLUSIONS

As mentioned in the introduction, there are many motivations for developing music generation systems. The main motivation for this work is a combination of artistic motivation, design of compositional tools and computational modelling of musical styles. The latter being examined in more detail in the final chapter on dance hit prediction. Although the music generated by the systems developed in this thesis is far from a complete composition, it could offer a source of inspiration to composers. Furthermore, many of the developed components and techniques offer useful tools to build a more complete composition system in future research. An overview of the contributions of this work is given below.

In Part 1 the focus lies on generating music based on rules from music theory. In Chapter 1 a variable neighbourhood search (VNS) algorithm is developed that can efficiently generate first species counterpoint music. The parameters of this algorithm are set based on a thorough statistical experiment. It is also shown that VNS outperforms both random search and a genetic algorithm. In order to evaluate the quality of the generated music, rules from music theory were quantified and used in the objective function. We worked with first-species counterpoint, since this is the most simplest form of counterpoint. Johan Fux designed the counterpoint system to teach students how to become composers, hence it makes sense to start from the first level, i.e., first species.

The algorithm is modified to generate fifth species counterpoint, a more complex form of music, in Chapter 2. A rhythmic aspect is now taken into account and the relevant rules from counterpoint theory are quantified for use in the objective function. A statistical experiment was performed to determine the optimal parameter settings. The resulting system is efficient and the generated music sounds pleasing to the subjective ear of the author.
CONCLUSIONS

In Chapter 3 the VNS algorithm was modified so that it can generate a continuous stream of music instead of one complete piece at a time. It was implemented as an Android application that instantly plays the generated music as it is being generated.

Part 2 allows us to break free from working with musical styles defined by music theory. Different methods are examined to use statistical models to construct quality measures for the generated music. Using machine learned models to capture a musical style allows us to work with a plenitude of styles and model aspects that might be to abstract to be included in rules from music theory, thus allowing us to generate more complete music. It might even stimulate the discovery of new rules and thus expand existing music theory.

In Chapter 4 different models that can classify between music composed by Bach, Beethoven and Haydn are learned from a corpus. Both comprehensible models, that are useful for theory building, and a more efficient black-box model are built. An efficient model is then implemented in the objective function combined with the previously defined rules for counterpoint. The Android application developed in Chapter 3 is modified to continuously generate counterpoint music with composer-specific characteristics.

A more complete statistical model is developed in Chapter 5. Whereby first species counterpoint is modelled by a first order Markov model based on vertical viewpoints. We worked with first species so that so that the model could be implemented in the objective function of the VNS from Chapter 1. It is then shown that VNS is more effective at generating high-probability sequences than both random walk and Gibbs sampling.

Chapter 6 builds on the previous chapter by exploring different ways in which a low order Markov model can be used to build quality assessment metrics that stimulate long term coherence. These are then evaluated in detail by comparing bagana music generated by the VNS with each of the metrics. We chose to work with music for bagana because it is such a simple form of music, on which the effect the different metrics can easily been seen by an expert. Secondly, it has a very specific repetitive structure. A method was developed that allows the generation of music into this cyclic structure. The types of generative models used in Chapter 5 and 6 seem to be much more complete
and more suited for music generation purposes versus classifier models such as the one used in Chapter 4.

In Part 2, Chapter 7 focusses on developing classification models. More specifically, applied to the task of predicting dance hits based on non-symbolic audio data. The results show that these models are able to differentiate between top 10 and lower positioned songs. This system is implemented as an online application where users can upload their audio data and get the probability of it being a hit.

The techniques developed in this thesis offer numerous possibilities for future research. It would be extremely interesting to work with other and more complex music, possibly consisting of more voices. Therefore, the move types of the VNS would need to be extended. More complex and higher order statistical models could be built and integrated in the objective function based on the multiple viewpoints method. A more complex structure, such as recurring themes with slight variations can be added. In the long run, the ideal would be that the techniques and algorithms stemming from this research are implemented in a robust way so that they can be applied to any type of polyphonic, multi-voiced music. This is however not an elementary task. Once a more complete piece of music is generated, an extensive experiment based on human evaluation of the pieces could provide valuable feedback about the implemented methods.
Er zijn vele motivaties om systemen te creëren die muziek genereren. De motivatie voor het maken van deze thesis is een combinatie van artistieke motivatie, het ontwikkelen van nieuwe compositie tools en het opstellen van computer modellen voor muzikale stijlen. Dit laatste wordt in meer detail onderzocht in het laatste hoofdstuk over dans hit voorspelling. Hoewel de muziek gegenereerd met de systemen uit deze thesis nog geen complete compositie is, kan het op dit moment mogelijks een inspiratiebron bieden voor componisten. Verder worden er nieuwe en efficiënte tools en technieken aangeboden die kunnen worden gebruikt voor het bouwen van een completer compositie systeem in toekomstig onderzoek.

In het eerste deel ligt de focus op het genereren van muziek gebaseerd op regels uit de muziektheorie. In Hoofdstuk 1 wordt een variable neighbourhood search algoritme (VNS) ontwikkeld dat op een efficiënte manier eerste vorm contrapunt kan genereren. De parameters van dit algoritme zijn bepaald door middel van een uitgebreid statistisch experiment. Er wordt aangetoond dat het VNS beter presteert dan zowel random walk als een genetisch algoritme. Om de kwaliteit van de muziek te evalueren werden regels uit de muziektheorie gekwantificeerd en geïmplementeerd in de doelfunctie.

Het algoritme is aangepast om vijfde vorm contrapunt te genereren, een complexere vorm van muziek, in Hoofdstuk 2. Er wordt nu rekening gehouden met een ritmisch aspect. De relevante regels uit de contrapunt theorie werden gekwantificeerd zodat ze kunnen worden gebruikt in de doelfunctie. Een statistisch experiment is uitgevoerd om de optimale parameter instellingen te bepalen. Het resulterende systeem is efficiënt en de gegenereerde muziek klinkt aangenaam, althans volgens de subjectieve mening van de auteur.
In Hoofdstuk 3 wordt het VNS aangepast zodat een continue stroom van muziek gegenereerd wordt in plaats van één compleet stuk. Dit is geïmplementeerd als een Android toepassing die de gegenereerde muziek meteen afspeelt tijdens het generatietransitieproces.

In Deel 2 wordt vrij gebroken van voorgedefinieerde muzikale stijlen door de muziektheorie. Verschillende generatiemethodes worden onderzocht in combinatie met statistisch modellen als kwaliteitsmaatstaven voor de gegenereerde muziek.

In Hoofdstuk 4 worden verschillende modellen geleerd van een corpus die kunnen classificeren tussen muziek gecomponeerd door Bach, Beethoven en Haydn. Sommige van deze modellen zijn begrijpbaar, zodat ze kunnen worden gebruikt voor het opstellen van nieuwe regels in de muziektheorie, andere modellen zijn meer black-box. Een van deze modellen is geïmplementeerd in de doelfunctie van het VNS algoritme, gecombineerd met de stijlregels voor contrapunt. De Android toepassing ontwikkeld in Hoofdstuk 3 is aangepast zodat een continue stroom van contrapunt muziek gegenereerd wordt met componist-specifieke kenmerken.

Een completer statistisch model wordt ontwikkeld in Hoofdstuk 5. Eerste vorm contrapunt wordt gemodelleerd door middel van een eerste orde Markov model gebaseerd op verticale viewpoints. Dit model wordt dan geïmplementeerd in de doelfunctie van het VNS algoritme. Er wordt aangetoond dat VNS efficiënter is in het genereren van sequenties met hoge probabiliteit dan random walk en Gibbs sampling.

Hoofdstuk 6 bouwt voort op het vorige hoofdstuk door de verschillende manieren te onderzoeken waarop een Markov model van lage orde kan worden geïntegreerd in een kwaliteitsmetriek die de coherentie op lange termijn stimuleert. Deze metrieken worden dan in detail bestudeerd door stukken baganamuziek te vergelijken die elk zijn gegenereerd met VNS en een specifieke metriek. De baganamuziek wordt gegenereerd in een zeer specifieke cyclische structuur.

In Deel 2 focus Hoofdstuk 7 op het ontwikkelen van classificatiemodellen voor het voorspellen van dans hits gebaseerd op niet-symbolische audio data.
Het resultaat van dit onderzoek toont aan dat deze modellen effectief kunnen differentiëren tussen een top 10 en een lager gepositioneerd nummer.

Ondanks het feit dat deze thesis een belangrijke bijdrage geleverd heeft aan het ontwikkelen van efficiënte methoden voor het genereren van muziek, is dit slechts een aanzet om tot een volledig automatisch gegenereerd muziekstuk te komen. Er zijn nog tal van interessante uitbreidingsmogelijkheden gebaseerd op de technieken gebruikt in dit onderzoek.
DETAILED BREAKDOWN OF OBJECTIVE FUNCTION
FOR FIRST SPECIES COUNTERPOINT

HORIZONTAL ASPECT OF THE OBJECTIVE FUNCTION

These rules apply to a tonal cantus firmus consisting of $L \times 16$ whole notes. $L$ is the length of the fragment expressed in units of 16 notes.

1. Notes should not be repeated immediately. In the case of counterpoint, maximum 4 tied notes per $L$ are allowed.

2. Only consonant horizontal intervals are permitted between consecutive notes. Horizontal intervals of 0, 1, 2, 3, 4, 5, 7, 8, 9, 12, 15 and 16 semitones are consonant. The largest allowed consonant interval between two consecutive notes is the perfect octave.

3. Conjunct: stepwise movement should predominate, 2 to 4 leaps per $L$ are optimal. Stepwise motion is an interval of 1 or 2 semitones. If the context is prevailing stepwise (90%), an exception can be made.

4. Each large leap should be followed by stepwise motion. A leap is an interval of more than 2 semitones. A large leap is an interval of more than 4 semitones.

5. Change direction after each large leap.

6. There should be one climax (highest note).

7. Climax should be melodically consonant with tonic.
8. No more than two large leaps per L.
9. No more than two consecutive leaps.
10. Do not move too long, stepwise, in same direction.
11. The direction needs to change several times (min 3 times per L).
12. The starting note should be the tonic. In the case of counterpoint, the starting note can also be the dominant.
13. The ending note should be tonic and in same octave as starting note.
14. The penultimate note should be the leading tone for CP and the supertonic for CF. The leading tone is the 7th note of the scale. In the case of a minor scale it should be raised by half a tone. When the previous note is the sixth note of the scale, it is also allowed to be raised by half a tone.
15. The beginning and the end of all motion should be consonant. A motion interval is the interval between the start and end note of an upward or downward movement.
16. There should be a good balance between ascending and descending motion. Large motion intervals (> 9 semitones) should be avoided.
17. No frequent repetition of a single note (maximum 3 times per L).
18. No repetition of sequences within a 16 note interval. A sequence is a group of at least two notes.

\[
subscore_1^H(s) = \begin{cases} 
\frac{\#\text{repeated notes}}{L} & \text{in case of CF and if } \#\text{repeated notes} \leq L, \\
1 & \text{in case of CF and if } \#\text{repeated notes} > L, \\
\frac{\#2 \text{repeated notes} - 2 \times L }{\text{length} - 2 \times L} & \text{in case of CP,}
\end{cases}
\]

\[
subscore_2^H(s) = 1 - \frac{\#\text{consonant intervals}}{\#\text{intervals}} \quad (A.1)
\]

\[
(A.2)
\]
**FIRST SPECIES RULES**

\[
\text{subscore}^H_3(s) = \begin{cases} 
0 
& \text{if } 2 \times L \leq \#\text{leaps} \leq 4 \times L, \\
\frac{\#\text{leaps} - (4 \times L)}{\text{length} - 1 - (4 \times L)} 
& \text{if } 4 \times L < \#\text{leaps}, \\
\frac{\#\text{leaps}}{\text{length} - 1 - (4 \times L)} 
& \text{if } 2 \times L > \#\text{leaps}. 
\end{cases}
\]  
(A.3)

\[
\text{subscore}^H_4(s) = \begin{cases} 
0 
& \text{if } \geq 90\% \text{ of intervals are } 1 \text{ or } 2 \text{ semitones}, \\
\frac{\#\text{large leaps not fol. by stepw. mot.}}{\#\text{large leaps}} 
& \text{if } < 90\% \text{ of int. are } 1 \text{ or } 2 \text{ semitones.} 
\end{cases}
\]  
(A.4)

\[
\text{subscore}^H_5(s) = \frac{\#\text{large leaps not followed by } \Delta \text{direction}}{\#\text{large leaps}} 
\]  
(A.5)

\[
\text{subscore}^H_6(s) = \frac{\#\text{occurences of highest note} - 1}{\text{length}} 
\]  
(A.6)

\[
\text{subscore}^H_7(s) = \begin{cases} 
0 
& \text{if climax is consonant with tonic,} \\
1 
& \text{if climax is not consonant with tonic.} 
\end{cases}
\]  
(A.7)

\[
\text{subscore}^H_8(s) = \begin{cases} 
0 
& \text{if } \#\text{large leaps} \leq 2 \times L, \\
\frac{\#\text{large leaps} - (2 \times L)}{\text{length} - 1 - (2 \times L)} 
& \text{if } \#\text{large leaps} > 2 \times L. 
\end{cases}
\]  
(A.8)

\[
\text{subscore}^H_9 = \frac{\# \text{ of } 3 \text{ consecutive leaps}}{\text{length} - 3} 
\]  
(A.9)

\[
\text{subscore}^H_{10}(s) = \begin{cases} 
0 
& \text{if longest stepwise sequence } \leq 5, \\
\frac{\text{length of longest stepwise sequence} - 5}{\text{length} - 6} 
& \text{if longest stepwise sequence } > 5. 
\end{cases}
\]  
(A.10)

\[
\text{subscore}^H_{11}(s) = \begin{cases} 
0 
& \text{if } \# \text{ direction changes } \geq 3 \times L, \\
1 - \frac{\# \text{ direction changes}}{3 \times L} 
& \text{if } \# \text{ direction changes } < 3 \times L. 
\end{cases}
\]  
(A.11)

\[
\text{subscore}^H_{12}(s) = \begin{cases} 
0 
& \text{if start note is tonic (or dominant in the case of CP),} \\
1 
& \text{if start note is not tonic.} 
\end{cases}
\]  
(A.12)
First Species Rules

\[ \text{subscore}_13^H (s) = \begin{cases} 0 & \text{if end note is tonic in same octave as start note,} \\ 0.5 & \text{if end note is tonic, but is not equal to start note,} \\ 1 & \text{if end note is not tonic.} \end{cases} \]  

(A.13)

\[ \text{subscore}_14^H (s) = \begin{cases} 0 & \text{if pen. is leading note (CP) or supertonic (CF) in same oct. as end note,} \\ 0.5 & \text{if pen. is leading note (CP) or supertonic (CF), not in same oct. as end note,} \\ 1 & \text{if pen. is not leading note (CP) or supertonic (CF).} \end{cases} \]  

(A.14)

\[ \text{subscore}_15^H (s) = \frac{\# \text{times that start and end of motion are dissonant}}{\# \text{motion intervals}} \]  

(A.15)

\[ \text{subscore}_16^H (s) = \frac{\# \text{motion intervals > 9 semitones}}{\# \text{motion intervals}} \]  

(A.16)

\[ \text{subscore}_17^H (s) = \frac{\# \text{notes repeated more than } (3 \times L) \text{ times}}{\text{length}} \]  

(A.17)

\[ \text{subscore}_18^H (s) = \frac{\# \text{notes that are part of a rep. seq. within a 16 note interval}}{\text{length}} \]  

(A.18)

Vertical Aspect of the Objective Function

1. Only consonant vertical intervals are permitted. Vertical consonance is determined by the tonal distance between the cantus firmus (CF) and the counterpoint (CP). Vertical intervals of 0, 3, 4, 7, 8, 9, 12, 15 and 16 semitones are consonant.

2. All perfect intervals should be approached by contrary or oblique motion. Intervals of 0, 5, 7 and 12 semitones are considered perfect. There are four types of motion:

   • Contrary motion: CP and CF move in a different direction.
First Species Rules

- Similar motion: CP and CF move in the same direction.

- Parallel motion: CP and CF move in the same direction at a constant (or virtually constant) distance. Thirds, sixths and tenths are considered parallel, even if the quality of the interval changes (minor vs major).

- Oblique motion: one voice remains stationary and the other voice moves.

3. Avoid the overlapping of parts. Overlap occurs when the CF is 1 or 2 semitones higher than the previous note of the CP. Or when the CP is 1 or 2 semitones lower than the previous note of the CF.

4. The maximum allowed distance between voices is one tenth (16 semitones). With the exception of the climax.

5. There should be no crossing. Crossing occurs when the lower voice is above the upper voice.

6. Unison is only allowed in the beginning and end.

7. Avoid leaps simultaneously in the cantus firmus and counterpoint. Especially leaps larger than one fourth (6 semitones) in the same direction.

8. Use all types of motion.

9. From unison to octave (and vice versa) is forbidden.

10. The ending should be unison or octave.

11. The beginning should be unison, octave or fifth above cantus.

12. Contrary motion should predominate slightly.

13. Thirds, sixths and tenths should predominate.

14. There can be no more than 3 successive parallel thirds, sixths and tenths.
15. The climax (note with the highest pitch) of the CF and CP should not coincide.

\[
\text{subscore}_1^V(s) = 1 - \frac{\#\text{consonant intervals}}{\#\text{intervals}} \quad (A.19)
\]

\[
\text{subscore}_2^V(s) = \begin{cases} 
\frac{\#\text{perfect intervals not appr. by contr. or obl. motion}}{\#\text{perfect intervals}} & \text{if } \#\text{perfect intervals} \neq 0, \\
0 & \text{if } \#\text{perfect intervals} = 0
\end{cases} \quad (A.20)
\]

\[
\text{subscore}_3^V(s) = \frac{\#\text{overlaps}}{\#\text{length-1}} \quad (A.21)
\]

\[
\text{subscore}_4^V(s) = \frac{\#\text{distances} > 16 \text{ semitones}}{\text{length}} \quad (A.22)
\]

\[
\text{subscore}_5^V(s) = \frac{\#\text{crossed notes}}{\text{length-1}} \quad (A.23)
\]

\[
\text{subscore}_6^V(s) = \frac{\#\text{unisons (excl beginning and end)}}{\text{length-2}} \quad (A.24)
\]

\[
\text{subscore}_7^V(s) = \frac{(\#\text{sim leaps not same dir}) + 2 \times (\#\text{sim leaps} > 6 \text{ semitones same dir})}{2 \times \#\text{intervals}} \quad (A.25)
\]

\[
\text{subscore}_8^V(s) = 1 - \frac{\#\text{different types of motion used}}{4} \quad (A.26)
\]

\[
\text{subscore}_9^V(s) = \frac{\#\text{motion from unison to octave and vsvs}}{\text{length}} \quad (A.27)
\]

\[
\text{subscore}_{10}^V(s) = \begin{cases} 
1 & \text{if end is not unison or octave}, \\
0 & \text{if end is unison or octave}.
\end{cases} \quad (A.28)
\]

\[
\text{subscore}_{11}^V(s) = \begin{cases} 
1 & \text{if beginning is not unison, octave or fifth above cantus}, \\
0 & \text{if beginning is unison, octave or fifth above cantus}.
\end{cases} \quad (A.29)
\]

\[
\text{subscore}_{12}^V(s) = \begin{cases} 
1 & \text{if } \#\text{ contrary motion} \leq \text{all other motion}, \\
0 & \text{if } \#\text{ contrary motion} > \text{all other motion}.
\end{cases} \quad (A.30)
\]
$$\text{subscore}_V^{13}(s) = \begin{cases} 0 & \text{if \# thirds, sixths and tenths } \geq 50\%, \\ 1 & \text{if \# thirds, sixths and tenths } < 50\%. \end{cases}$$ \tag{A.31}$$

$$\text{subscore}_V^{14}(s) = \frac{\# \text{sequences of parallel thirds, sixths and tenths } \geq 3}{\text{length-3}} \tag{A.32}$$

$$\text{subscore}_V^{15}(s) = \begin{cases} 1 & \text{if the climax of the CF and CP coincide,} \\ 0 & \text{if the climax of the CF and CP do not coincide.} \end{cases} \tag{A.33}$$
DETAILED BREAKDOWN OF THE OBJECTIVE FUNCTION FOR FIFTH SPECIES COUNTERPOINT

These rules are valid for a fifth species counterpoint fragment of $L \times 16$ measures. $L$ can be seen as the length of the music, expressed in units of 16 measures.

**HORIZONTAL ASPECT OF THE OBJECTIVE FUNCTION**

- Eight notes (8ths) must move in step.
- There should be one climax, that can only be repeated after a neighbouring tone. A neighbouring note is positioned between two repeated notes and is approached and left by stepwise motion. The two repeated notes must always be vertically consonant.
- The climax falls on a strong beat, except when it is repeated after a neighbouring tone.
- Only consonant horizontal intervals are permitted. Horizontal intervals of 0, 1, 2, 3, 4, 5, 7, 8, 9 and 12 semitones (or more octaves) are consonant.
- Conjunct: stepwise movement should predominate. Two to four leaps per $L$ are optimal. Stepwise motion is an interval of one or two semitones.
- Each large leap should be followed by stepwise motion. A *leap* is an interval of more than two semitones. A *large leap* is an interval of more than 4 semitones.
Fifth species rules

- Change direction after each large leap.
- The climax should be melodically consonant with tonic.
- No more than two consecutive leaps.
- No more than two large leaps per L.
- Do not move too long stepwise in same direction.
- The direction needs to change several times (min. three times per L).
- The ending note should be tonic.
- The penultimate note should be the leading tone for CP and the supertonic for CF. The leading tone is the 7th note of the scale. In the case of a minor scale it should be raised by half a tone. Preferably in the same octave as the ending note.
- The beginning and the end of all motion should be consonant. A motion interval is the interval between the start and end note of an upward or downward movement.
- There should be a good balance between ascending and descending motion. Large motion intervals (> 9 semitones) should be avoided.
- Maximum four tied notes per L are allowed.
- No repetition of sequences within a 16 note interval. A sequence is a group of at least two notes.
- The largest allowed interval between two consecutive notes is the perfect octave.
Fifth Species Rules

\[ \text{subscore}_1^H(s) = \frac{\#8\text{ths not preceded by step} + \#8\text{ths not left by step}}{\#8\text{ths} \times 2} \]  
\[ \text{(B.1)} \]

\[ \text{subscore}_2^H(s) = \frac{\#\text{highest note not as neighb. tone} - 1}{\text{length}} \]  
\[ \text{(B.2)} \]

\[ \text{subscore}_3^H(s) = \frac{\#\text{highest notes on weak beat (except after neighb. tones)}}{\#\text{highest notes}, \text{excl. those after neighb. tones}} \]  
\[ \text{(B.3)} \]

\[ \text{subscore}_4^H(s) = 1 - \frac{\#\text{consonant intervals}}{\#\text{intervals}} \]  
\[ \text{(B.4)} \]

\[ \text{subscore}_5^H(s) = \begin{cases} 
0 & \text{if } 2 \times L \leq \# \text{leaps} \leq 4 \times L, \\
\frac{\#\text{leaps} - (4 \times L)}{\text{length} - 1 - (4 \times L)} & \text{if } 4 \times L < \# \text{leaps}, \\
\frac{\#\text{leaps}}{\text{length} - 1 - (4 \times L)} & \text{if } 2 \times L > \# \text{leaps}.
\end{cases} \]  
\[ \text{(B.5)} \]

\[ \text{subscore}_6^H(s) = \frac{\#\text{large leaps not foll. by stepw. mot.}}{\#\text{large leaps}} \]  
\[ \text{(B.6)} \]

\[ \text{subscore}_7^H(s) = \frac{\#\text{large leaps not followed by } \Delta \text{direction}}{\#\text{large leaps}} \]  
\[ \text{(B.7)} \]

\[ \text{subscore}_8^H(s) = \begin{cases} 
0 & \text{if climax is consonant with tonic,} \\
1 & \text{if climax is not consonant with tonic.}
\end{cases} \]  
\[ \text{(B.8)} \]

\[ \text{subscore}_9^H = \frac{\# \text{of 3 consecutive leaps}}{\text{length} - 3} \]  
\[ \text{(B.9)} \]

\[ \text{subscore}_{10}^H(s) = \begin{cases} 
0 & \text{if } \#\text{large leaps} \leq 2 \times L, \\
\frac{\#\text{large leaps} - (2 \times L)}{\text{length} - 1 - (2 \times L)} & \text{if } \#\text{large leaps} > 2 \times L.
\end{cases} \]  
\[ \text{(B.10)} \]

\[ \text{subscore}_{11}^H(s) = \begin{cases} 
0 & \text{if longest stepwise seq.} \leq 5, \\
\frac{\text{length of longest stepwise seq.}}{\text{length}} & \text{if longest stepwise sequence} > 5.
\end{cases} \]  
\[ \text{(B.11)} \]

\[ \text{subscore}_{12}^H(s) = \begin{cases} 
0 & \text{if } \# \text{ direction changes} \geq 3 \times L, \\
1 - \frac{\#\text{direction changes}}{3 \times L} & \text{if } \# \text{ direction changes} < 3 \times L.
\end{cases} \]  
\[ \text{(B.12)} \]
FIFTH SPECIES RULES

\[
subscore_{13}^H(s) = \begin{cases} 
0 & \text{if end note is tonic,} \\
1 & \text{if end note is not tonic.}
\end{cases} \tag{B.13}
\]

\[
subscore_{14}^H(s) = \begin{cases} 
0 & \text{if pen. is leading note (CP) or supertonic (CF) in same oct. as end,} \\
0.5 & \text{if pen. is leading note (CP) or supertonic (CF), not in same oct. as end note,} \\
1 & \text{if pen. is not leading note (CP) or supertonic (CF).}
\end{cases} \tag{B.14}
\]

\[
subscore_{15}^H(s) = \frac{\text{#times that start and end of motion are dissonant}}{\text{#motion intervals}} \tag{B.15}
\]

\[
subscore_{16}^H(s) = \frac{\text{#motion intervals > 9 semitones}}{\text{#motion intervals}} \tag{B.16}
\]

\[
subscore_{17}^H(s) = \frac{\text{#notes repeated more than } (2 \times L) \text{ times}}{\text{length}} \tag{B.17}
\]

\[
subscore_{18}^H(s) = \frac{\text{#notes part of a rep. seq. within a 4 measure interval}}{\text{length}} \tag{B.18}
\]

\[
subscore_{19}^H = \frac{\text{#intervals > 12 semitones}}{\text{#intervals}} \tag{B.19}
\]

VERTICAL ASPECT OF THE OBJECTIVE FUNCTION

- Whole notes should always be vertically consonant.
- Half notes should always be consonant on first beat, unless they are suspended and continued stepwise and downward. They can be dissonant on second beat if they are a passing tone.
- Quarter notes should always be consonant on the first beat, unless they are suspended and continued stepwise and downward. They can be dissonant on other beats if they are a passing tone or a neighbouring tone.
- Eight notes occur in pairs - at least one of them should be vertically consonant.

- Unisons are allowed on the 1st beat through suspension if they are tied over or followed by stepwise motion.

- The maximum allowed distance between voices is 16 semitones, except on the climax.

- There should be no crossing.

- Avoid the overlapping of parts.

- From unison to octave (and vice versa) is forbidden.

- The ending should be a unison or octave.

- All perfect intervals (also perfect 4th) should be approached by contrary or oblique motion. Intervals of 0, 5, 7 and 12 semitones are considered perfect.

- Avoid leaps sim. cf and cp, especially large leaps (> 6 semitns) in the same direction.

- Use all types of motion.

- The climax of the CF and CP should not coincide.

- Successive unisons, octaves and fifths on first beats are only valid when separated by three quarter notes.

- Successive unisons, octaves and fifths not on first beats are only valid when separated by at least two notes. For afterbeat fifths and octaves only separated by a single quarter this is allowed in the case of a consonant suspension of a quarter note.

- No ottava or quinta battuda.
Fifth Species Rules

- Best ending is a dissonant suspension into the leading tone. This can be decorated.
- Thirds, sixths and tenths should predominate.

\[
\begin{align*}
\text{subscore}_1^V (s) &= \frac{\# \text{dissonant whole notes}}{\# \text{whole notes}} \\
\text{subscore}_2^V (s) &= \frac{\# \text{dissonant half notes exc. passing tones or weak beats}}{\# \text{half notes}} \\
\text{subscore}_3^V (s) &= \frac{\# \text{dissonant quarters exc. passing and neighb. tones or weak beats}}{\# \text{quarters}} \\
\text{subscore}_4^V (s) &= \frac{\# \text{pairs of eight notes with both notes dissonant}}{\# \text{pairs of eight notes}} \\
\text{subscore}_5^V (s) &= \frac{\# \text{non-allowed unisons}}{\# \text{notes}} \\
\text{subscore}_6^V (s) &= \frac{\# \text{distances > 16 semitones}}{\text{length}} \\
\text{subscore}_7^V (s) &= \frac{\# \text{crossed notes}}{\text{length}} \\
\text{subscore}_8^V (s) &= \frac{\# \text{overlaps}}{\text{length} - 1} \\
\text{subscore}_9^V (s) &= \frac{\# \text{motion from unison to octave and vsvs}}{\# \text{intervals}} \\
\text{subscore}_{10}^V (s) &= \begin{cases} 0 & \text{if the ending is unison or octave}, \\ 1 & \text{otherwise}. \end{cases} \\
\text{subscore}_{11}^V (s) &= \frac{\# \text{perfect intervals not appr. by contr. or obl. motion}}{\# \text{intervals}} \\
\text{subscore}_{12}^V (s) &= \frac{\# \text{sim leaps not same dir} + (\# \text{sim leaps > 6 semit. same dir})}{2 \times \# \text{intervals}}
\end{align*}
\]
Fifth Species Rules

\[ \text{subscore}_{13}^V(s) = 1 - \frac{\text{#different types of motion used}}{4} \]  \hspace{1cm} (B.32)

\[ \text{subscore}_{14}^V(s) = \begin{cases} 
1 & \text{if the climax of the CF and CP coincide}, \\
0 & \text{if the climax of the CF and CP do not coincide}. 
\end{cases} \]  \hspace{1cm} (B.33)

\[ \text{subscore}_{15}^V(s) = \frac{(\text{#succ. unis., 8ths or 5ths on first beats not sep. by 3 quarters})}{\#\text{measures} - 1} \]  \hspace{1cm} (B.34)

\[ \text{subscore}_{16}^V(s) = \frac{(\text{#succ. unis., 8ths or 5ths not first beats not sep. by 2 unsusp.n.})}{\#(\text{notes} - \text{measures})} \]  \hspace{1cm} (B.35)

\[ \text{subscore}_{17}^V(s) = \frac{\text{#ottava and quinta battudas}}{\#\text{measures}} \]  \hspace{1cm} (B.36)

\[ \text{subscore}_{18}^V(s) = \begin{cases} 
1 & \text{if ending is dissonant suspension into the leading tone (can be decorated)}, \\
0.5 & \text{if ending is a suspension}, \\
0 & \text{otherwise}. 
\end{cases} \]  \hspace{1cm} (B.37)

\[ \text{subscore}_{19}^V(s) = \begin{cases} 
0 & \text{if # thirds, sixths and tenths } \geq 50\%, \\
1 - \frac{2 \times \text{# thirds, sixths and tenths}}{\#\text{notes}} & \text{otherwise}. 
\end{cases} \]  \hspace{1cm} (B.38)
LIST OF PUBLICATIONS

Journal articles


D. Herremans, K. Sørensen and D. Conklin. 2014. **Sampling the extrema from statistical models of music with neighbourhood search.** *Proceedings of ICMC–SMC.* Athens. 1096–1103.


C. J. Bates D. The r project for statistical computing, 5 2012. URL http://www.r-project.org/.


BI-BLIOGRAPHY


P. Yong-Cai, L. Wen-chao, and L. Xiao. Development and research of music player application based on android. In Communications and Intelligence

INDEX

n-gram model, 83
n-grams, see n-gram model

ACE XML file, 88
Analysis of Variance, 30, 32, 43, 59, 62
Android, 68, 71
Android Native Development Kit, 72
Android Software Development Kit, 72
ANOVA, see Analysis of Variance
ant colony optimization, 12, 13, 126
antipattern, 138, 139, 149
area under the receiver operating curve, 166, 170, 171, 173, 176, 177, 181
area under the receiver operator curve, 179
AUC, see area under the receiver operating curve

bagana, 126, 128–131, 138, 141, 142, 146
BB, see Billboard
Billboard, 157, 158, 161
binary tournament selection, 42
black-box model, 16, 20, 84, 90

C4.5 algorithm, 84, 88, 90, 93–95, 170

CAC, see computer aided composition
cantus firmus, 14, 17, 22, 30, 51, 54, 58, 59, 69, 70, 73, 74, 76, 110, 112, 119, 191, 194
CF, see cantus firmus
classification model, 82, 84, 86, 88–90, 104, 155, 158, 163, 166–168, 176
combinatorial optimization problem, 12, 115, 140
composer classification, 83, 88, 91
compound pattern, 131, 132
comprehensible model, 84, 90, 93, 95
counterpoint, 12, 14–17, 52, 69, 102, 112
fifth species, 51, 53, 199
first species, 17, 51, 111, 194
rules, 17, 18, 52–54
CP, see counterpoint
crisp classification, 95
cross-entropy, 116, 118–121, 134–136, 142–146, 148
cross-validation, 90, 91, 135, 146, 170, 176, 177, 179
cyclic pattern, 128, 131, 132, 149
Dalvik Virtual Machine, 72
DE, see delta cross-entropy
decision tree, 82, 84, 90, 93, 94, 168, 170, 171, 177
delta cross-entropy, 135, 146
dyad, 112–116, 118, 119, 123
dynamic programming, 110, 116, 133
evolutionary algorithm, 10, 13–17, 20, 42, 44, 46, 52, 55, 126
false negative rate, 179
feature selection, 167, 168, 177, 181
first order, 115, 126–128, 133, 142
FNR, see false negative rate
FuX, 68–70, 76–78, 102–104
Fux, see Fux, Johann
Fux, Johann, 15, 17, 52, 53
GA, see genetic algorithm
genetic algorithm, see evolutionary algorithm
genre classification, 83, 155, 160, 166
Gibbs sampling, 110, 115, 118, 120, 121
grammars, 15, 83, 110
GridSearch, 99, 176
GS, see Gibbs sampling
hard constraints, 19, 55, 119, 139
harmonic rules, 14, 17, 54, 69, 102
high order, 127
high probability sequence, 111, 115, 116, 123, 133, 134, 137, 140, 143, 147
high probability solution, see high probability sequence
human fitness bottleneck, 13, 52, 110, 126
information contour, 137, 141, 142, 147, 148
J48, 93, 170
jMIRUtilities, 88
JRip, 91, 170
jSymbolic, 86, 87
K-means clustering, 84
c kernel, 176
 polynomial, 176, 178, 179
 RBF, 99, 176, 178, 179
KernScores, 86
logistic regression, 82, 89, 90, 95–98, 103, 173, 177, 179, 181
long term coherence, 126, 128, 139, 150
low order, 127, 128, 150
machine learning, 83, 84, 89, 122, 166
Markov chain, see Markov Model
maximum likelihood sequence, see high probability sequence
melodic rules, 14, 17, 54, 69, 102
metaheuristics, 12, 16, 20, 59, 69, 126
  constructive, 12, 13
  local search, 12, 13, 16, 21, 46, 50, 55, 132
  population-based, 13, 55
MIDI, 57, 72–74, 85–87, 112
MIR, see Music Information Retrieval
motion chart, 161, 162
Multi-Way ANOVA, see Analysis of Variance
multiple viewpoint method, 111, 123, 149
MuseScore, 28, 57, 58
Music Information Retrieval, 82, 155
  audio, 85
  symbolic, 85
MusicXML, 28, 57
naive Bayes, 84, 172, 177
NDK, see Android Native Development Kit
nearest neighbour, 84
neural network, 16, 47, 84, 89, 96, 173
normalization, 167
objective function, 18, 19, 54, 69, 100, 116, 134–137, 139, 191, 199
OCC, see Official Charts Company
Official Charts Company, 156, 158, 161, 163
Optimuse, 10, 28, 50–54, 57, 68, 70
out-of-time test set, 177, 181
overfitting, 88, 96, 173
Palestrina, 84, 110
predictive model, 88
probabilistic logic, 15
probabilistic method, 126
R, software, 30, 59
random search, 16, 29, 40, 41, 44, 46
random walk, 110, 115, 118–120, 122, 123, 127, 128, 132, 133, 139
receiver operating curve, 179
repeated incremental pruning to produce error reduction algorithm, 88, 90, 91, 170
RIPPER, see repeated incremental pruning to produce error reduction algorithm
ROC, see receiver operating curve
rule extraction, 90
rule induction, 90
rule-based system, 11, 16
ruleset, 82, 90, 170
RW, see random walk
sampling methods, 111, 115, 119, 123, 132
SDK, see Android Software Development Kit
sequential minimal optimization, 174
shifting perceptron model, 156
SimpleLogistic, 95, 173
simulated annealing, 69, 134
SMO, see sequential minimal optimization
soft constraints, 19, 54
spectral flux, 85
supervised learning algorithm, 89
support vector machines, 90, 98, 100, 155, 174, 176, 177
SVM, see support vector machines
temporal features, 160, 182
test set, 84, 177, 179, 181
timbre, 14, 160, 162, 170, 171
TM, see transition matrix
TPR, see true positive rate
training set, 91, 172, 174
transition matrix, 115, 116, 142
true positive rate, 179
unword, 133, 138, 139, 142, 146, 147, 149
variable neighbourhood search, 21, 22, 26, 55, 69, 82, 102, 115, 116, 126, 140
adaptive weights mechanism, 25, 56, 62
local optimum, 21
move type, 23
neighbourhood, 21, 23, 56
perturbation, 24, 56, 60
steepest descent, 24, 56
tabu list, 25, 56
tabu tenure, 26, 60
vertical viewpoints method, 111–113, 115, 123
Viterbi, 111, 115, 116, 119–121
VNS, see variable neighbourhood search
Weka, 88, 89, 91, 93, 95, 166–168, 170, 173, 174, 176
zero-crossing rate, 85