Setting Optimal Parameters on a VNS for Music Generation

D. Herremans
Advanced optimization using metaheuristics, April 10th, 2014
Overview

Generating counterpoint music
  Counterpoint?
  Variable Neigborhood Search

Setting parameters
  Setup & getting your data
    JMP
    R
  Results

Comparing to other algorithms

Conclusion
Computer aided composing (CAC)

Composing music = combinatorial optimization problem
Computer aided composing (CAC)

Composing music = combinatorial optimization problem

- Music → combination of notes
- “Good” music → fits a style as well as possible
- Formalized and quantified “rules” of a style → objective function
Counterpoint

- Polyphonic baroque music
- Inspired Bach, Haydn,…
- One of the most formally defined musical styles
  → Rules written by Fux in 1725
1st species counterpoint

- Counterpoint & Cantus firmus

\[
\begin{array}{cccccccccccccccc}
60 & 65 & 64 & 62 & 60 & 64 & 65 & 67 & 67 & 69 & 62 & 64 & 64 & 60 & 59 & 60 \\
\end{array}
\]

- Represented as 2 vectors with midi values
5th species counterpoint

- Counterpoint & Cantus firmus

- Represented as a vector of note objects, each with:
  - Pitch: midi value
  - Duration
  - Beat number
  - Measure number
  - Tied?
Quantifying musical quality

Examples of rules:

▶ Each large leap should be followed by stepwise motion in the opposite direction
▶ Half notes should always be consonant on the first beat, unless they are suspended and continued stepwise and downward
▶ All perfect intervals should be approached by contrary or oblique motion

→ 19 vertical and 19 horizontal subscores between 0 and 1
Quantifying musical quality

- Eight notes (8ths) must move in step.

\[ subscore^H_1(s) = \frac{\#8ths \text{ not preceded by step} + 8ths \text{ not left by step}}{\#8ths \times 2} \]  

(1)

- Whole notes should always be vertically consonant.

\[ subscore^V_1(s) = \frac{\#\text{dissonant whole notes}}{\#\text{whole notes}} \]  

(2)
Quantifying musical quality

\[ f_{cf}(s) = \sum_{i=0}^{19} a_i \cdot \text{subscore}_{cf_i}^H(s) \]  \hspace{1cm} (3)

horizontal aspect

\[ f_{cp}(s) = \sum_{i=0}^{19} a_i \cdot \text{subscore}_{cp_i}^H(s) + \sum_{j=0}^{19} b_j \cdot \text{subscore}_{j}^V(s) \]  \hspace{1cm} (4)

horizontal aspect \hspace{2cm} vertical aspect

\[ f(s) = f_{cf}(s) + f_{cp}(s) \]  \hspace{1cm} (5)
Quantifying musical quality

- Weights $a_i$ and $b_j$
- Specified at input
  - Emphasize subscore from start
- Adaptive weights mechanism
  - Increase weight of subscore with highest value
  - Keeps the search in the right direction
Variable Neighborhood Search

- Local search with 3 neighborhoods
- Selection
  - Steepest descent
  - Based on adaptive score $f^a(s)$

<table>
<thead>
<tr>
<th>$N_i$</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_{sw}$</td>
<td>Swap</td>
<td>Swap two notes</td>
</tr>
<tr>
<td>$N_{c1}$</td>
<td>Change1</td>
<td>Change one note</td>
</tr>
<tr>
<td>$N_{c2}$</td>
<td>Change2</td>
<td>Change two notes</td>
</tr>
</tbody>
</table>
Variable Neighborhood Search

- Excluded fragments
  - Tabu list
  - Infeasible
- Perturbation
  - Change r% of the notes randomly
- Adaptive weights mechanism
- Update best solution $s_{\text{best}}$, based on original score $f(s_{\text{best}})$
Setting parameters

- Are all elements really contributing?
- How do we set their parameters?

→ What needs to be tested?
Setting parameters

- Full factorial experiment, n=2304

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
<th>Nr. of levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_{sw}$ - Swap</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>$N_{c1}$ - Change1</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>$N_{c2}$ - Change2</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>Random move</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. iterations</td>
<td>5, 20, 50</td>
<td>3</td>
</tr>
<tr>
<td>Length of music</td>
<td>16, 32 measures</td>
<td>2</td>
</tr>
</tbody>
</table>
What to compare

- Objective function & $f(s)$
- Time:
  - System independent, e.g. number of $f(s)$ calculations
  - User time
```c
#include <time.h>
#include <sys/time.h>
#include <sys/times.h>
...

struct tms start_time;
struct tms end_time;

times(&start_time); // at start

times(&end_time); // at end

final=end_time.tms_utime- start_time.tms_utime;
cout << ((double)final / ((double)CLOCKS_PER_SEC))
```
Flipping switches in your code

if (nbh1 == true){
    //execute LS in neighbourhood of type 1
}

vector<int> tabulist1(tabulength1);

→ Don’t hard code.
Passing command line arguments

```c
>> myprogram -nbh1 0 -randsize 5 ...

int main(int argc, char *argv[]) {

    //set default values
    nh1 = true;
    randsize = 10;

    //read in the command line values

}
Passing command line arguments

```c
int i = 1;
while (i < argc) {
    string a = argv[i];
    if (argv[i][0] == '-') {
        string b = argv[i + 1];
        if (a == "-randsize") {
            randsize = atoi(b.c_str());
        } else if (a == "-nbh1") {
            if (b == "0") {
                nbh1 = false;
            }
        } i+=2;
    }
```

Running all combination with a bash script

>> nohup experiment.sh&

for randsize in 0 4 8; do
for nbh1 in 0 1; do
#run
../optimuse -randsize $randsize -nbh1 $nbh1
done
done
Coping with long runtime

- Parallelize runs
  - Split up per instance, or nbh, . . .
  - Use a parallelization script
- Use nohup . . . &
- Split up in two experiment with unrelated parameters
- Design of experiments
### Example output

<table>
<thead>
<tr>
<th>nh1</th>
<th>nh2</th>
<th>nh3</th>
<th>randsize</th>
<th>aweights</th>
<th>tt1</th>
<th>tt2</th>
<th>tt3</th>
<th>iters</th>
<th>length</th>
<th>score</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0.0</td>
<td>0.28</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0.07</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0.07</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0.244755</td>
<td>0.14</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0.06</td>
</tr>
</tbody>
</table>
Fitting a model
Loading data
Basic linear model, $R^2$
Interaction effects
Random effects
Mean plots
Profiler
Interaction plots
R script - reading in data

expdata<-read.table("filename.csv", header=TRUE)
names(expdata)<-c('nh1','randsize','tos','time')
attach(expdata)

nh1<-factor(nh1)
randsizename<-factor(randsize)
R script - fitting a linear model

//linear model
fit<-lm(tos ~ nh1 + randsize )

//with interaction effects
fit<-lm(tos ~ nh1 * randsize )

//mixed model with a random effects
fit<-lmer(tos ~ nh1 + randsize + (1 | instance) )

summary(fit)
anova(fit)

interaction.plot(nh2, nh3, tos)

→ conjugateprior.org/2013/01/formulae-in-r-anova/
Experiments & Results

- Multi-Way ANOVA model with interaction effects, using R
- $R^2 = 0.98$

<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>3909.2959</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>0.1048791</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>
</tbody>
</table>
Experiments & Results

- Mean plot for the size of the random jump

![Graph showing the relationship between random size (in %) and score over time (s).]
## Optimal parameter settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_{sw}$</td>
<td>on with $tt_{sw} = \frac{1}{16}$</td>
</tr>
<tr>
<td>$N_{c1}$</td>
<td>on with $tt_{c1} = \frac{1}{16}$</td>
</tr>
<tr>
<td>$N_{c2}$</td>
<td>on with $tt_{c2} = \frac{1}{16}$</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>50</td>
</tr>
</tbody>
</table>
Visualising performance
Comparing with other algorithms

(b) 

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Objective function CP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Search</td>
<td></td>
</tr>
<tr>
<td>VNS</td>
<td></td>
</tr>
<tr>
<td>GA</td>
<td></td>
</tr>
</tbody>
</table>

Time (seconds)

Objective function CP vs. time (seconds)

# of evaluated solutions

Objective function CP vs. # of evaluated solutions
Comparing with other algorithms
Comparing with other algorithms

- Uniform stopping criteria (stagnation – user time –...)
- Classical non-parametrical tests on population means:
  - One-sided Mann-Whitney-Wilcoxon \((k = 2)\)
  - Tukey-Duckworth \((k \geq 2)\)
  - Friedman \((k \geq 2\) with \(b\) instances)
Results

Example of a generated fragment with score 0.556776.
Conclusion

Always test your parameters and compare your algorithm to others if possible.

Keeping in mind:

- Random effects
- Interaction between factors
- Correct time reference
- Visualisation
Setting Optimal Parameters on a VNS for Music Generation

D. Herremans
Advanced optimization using metaheuristics, April 10th, 2014

dorien.herremans@uantwerpen.be