factorOracle: an Extensible Max External for Investigating Applications of the Factor Oracle Automaton in Real-Time Music Improvisation

Adam James Wilson
CUNY New York City College of Technology

Recommended Citation

This Article is brought to you for free and open access by the New York City College of Technology at CUNY Academic Works. It has been accepted for inclusion in Publications and Research by an authorized administrator of CUNY Academic Works. For more information, please contact AcademicWorks@cuny.edu.
factorOracle: an Extensible Max External for Investigating Applications of the Factor Oracle Automaton in Real-Time Music Improvisation

Adam James Wilson
awilson@citytech.cuny.edu
Emerging Media Technology Program, Department of Entertainment Technology
New York City College of Technology, City University of New York
Brooklyn, NY

Abstract
There are several extant software systems designed to generate music in real-time using a factor oracle automaton constructed from the musical input of a human improvisor. The impetus for the design of the factorOracle external is neither a desire to supersede these systems nor introduce novel algorithms for traversing the oracle, but rather to provide a fast, canonical interface for the automaton in Cycling74’s Max and, in future iterations, the Pure Data programming environment. Technical features of the factorOracle software are introduced here.

Background
The factor oracle is a directed acyclic word graph capable of expressing at least all of the substrings of a given word. The oracle can be built incrementally, in O(m) with respect to both processing time and memory consumption (Allauzen, Crochemore, and Raffinot 1999). The input can be a chain of pitches, durations, notes, collections of audio samples, or other musical features. The factorOracle external (1) continuously builds the oracle for a set of incoming musical data, and (2) generates variations on the stored data by exploiting features of the oracle that identify repeated patterns in the input.

Algorithm 1 Function addLetter(Oracle(\(p = p_1 \ p_2 \ldots \ p_m\), \(\sigma\)) (Allauzen, Crochemore, and Raffinot 1999)
1: Create a new state \(m + 1\)
2: Create a transition \(\sigma\) from \(m\) to \(m + 1\)
3: \(k \leftarrow S_p(m)\)
4: while \(k > -1\) and there is no transition \(\sigma\) from \(k\) do
5: \(\quad\) Create a new transition \(\sigma\) from \(k\) to \(m + 1\)
6: \(\quad\) \(k \leftarrow S_p(k)\)
7: \(\) end while
8: if \(k == -1\) then
9: \(\quad\) \(s \leftarrow 0\)
10: else
11: \(\quad\) \(s \leftarrow\) the state reached by transition \(\sigma\) from \(k\)
12: end if
13: \(S_{p\sigma}(m + 1) \leftarrow s\)
14: return Oracle(\(p = p_1 \ p_2 \ldots \ p_m\sigma\))

Algorithm 2 Function buildOracle(\(p = p_1 \ p_2 \ldots \ p_m\)) (Allauzen, Crochemore, and Raffinot 1999)
1: Create Oracle(\(\epsilon\)) with a single state 0
2: \(S_p(0) \leftarrow -1\)
3: for \(i \leftarrow 1\) to \(m\) do
4: \(\quad\) Oracle(\(p = p_1 \ p_2 \ldots \ p_i\)) \leftarrow addLetter(Oracle(\(p = p_1 \ p_2 \ldots \ p_{i-1}, p_i\))
5: end for

For each new transition \(\sigma\) and state \(m + 1\), we follow the suffix link for state \(m\) back to an earlier state \(k\) (this action describes the supply function \(S_p()\) shown in Algorithm 1). If there is no transition \(\sigma\) from the earlier state to the new state, we add it. We continue to follow the suffix links back and add transition \(\sigma\) when needed, until we reach state \(-1\) or a state \(k_{final}\) that has a transition \(\sigma\) to some other state. In the

This work is licenced under Creative Commons “Attribution 4.0 International” licence, the International Workshop on Musical Metacreation, 2016, (www.musicalmetacreation.org).
former case, we add a suffix link from $m + 1$ to state 0, and in the latter we add a suffix link from $m + 1$ to the state pointed
to by $\sigma$ from $k_{\text{final}}$.

It is important to note that all of the systems that use
the factor oracle to generate music turn the oracle into a
cyclic graph, allowing navigation via suffix links or reverse
suffix links (Assayag et al. 2006; Assayag and Bloch 2007;
Cont, Assayag, and Dubnov 2007; Dubnov and Surges 2013;
Dubnov and Wang 2014; Dubnov, Hsu, and Wang 2015).
When both reverse and forward suffix links are used, the
graph also becomes partially undirected, violating another
constraint of the originally proposed structure. Storage of
suffix links for use post-construction increases the memory
footprint of the automaton, which was developed in part as a
means of data compression.

Suffix links, however, appear to be the most important fea-
ture of the oracle for recombination of musical data, since
they connect the ends of substrings that have the longest re-
peated suffix with respect to the substring ended by the origin
state of the suffix link. The smoothness of the shift between
musical passages depends upon the length of the merged seg-
ment (longer == smoother). A method for determining the
length of the longest repeated suffix is described in (Lefebvre
and Lecroq 2000), and is employed in the generative systems
previously cited.

Since the oracle is built incrementally and in real time, it
is ideal for alternation between learning structure and gen-
erating music (Assayag et al. 2006), and can therefore be
developed as an artificial improvisation partner, beginning
with the modifications described above. Prior systems have
been designed to work with both symbolic music and audio
data; these employ various methods for identifying informa-
tion of interest in the oracle. Instantaneous style reproduction
and pattern matching are the primary applications.

Rationale

The software available for generating music with the factor
oracle seems to require older or reducible technological de-
pendencies. OMax is a collection of nine externals, which
appear to be unsupported for Max versions 6 and above (Levy
2012), and PyOracle’s real-time component apparently re-
quires embedding Python in Max (Dubnov and Surges 2013).
The factorOracle external (see the new alpha-phase source
code1) came about from a desire for an implementation that
is:

- fast—written purely in C, as opposed to Java or an inter-
preted language
- simple in function and design
- integrated with one or more of the most popular interactive
real-time audio environments (Max, Pure Data) as a single
external
- usable by a non-expert without modification or extension
- not beholden to any particular generative or analytical
model, but extensible by an expert user through patching
or source code modification

1https://github.com/ajwnyct/factorOracle

Figure 2: Max patch showing all factorOracle I/O options.

Construction

Data used to build the oracle are sent in the left inlet of the
external (see Figure 2). Older versions of the software sup-
ported multiple data types: symbols, floats, integers, and lists
(Wilson 2009; Brent and Wilson 2012). In this version, input
is limited to integers, reducing code maintenance overhead
and increasing performance. The onus is on the user to pro-
vide, when necessary, a suitable encoder/decoder to translate
information going into and coming out of the oracle; poly-
phonic musical data, for instance, must be organized into
meaningful structures reconstituted as integers.

In a recent performance incorporating the factorOracle
external (see Figure 3), MIDI information was mapped to
a matrix of 19 [0-18] equal-tempered pitch classes by 24
[0-23] durations. Each pitch/rhythm pair ($p, r$) was assigned
a cross-alphabet letter $c$ by $c = (p + 24 + r)$, and decoded by
$r = (c \mod 24), p = (\lfloor c/24 \rfloor \mod 19)$ (Artinian and Wilson
2016). A cross-alphabet of any dimensionality could be sim-
ilarly expressed, provided the total number of cells in the
matrix is less than the largest representable integer. For ex-
ample, a multidimensional matrix of audio features could be
sectioned into a finite number of “geographical” regions, and
analysis frames from an input signal could be assigned num-
bers corresponding to their locations within the matrix. Of
course, establishing the resolution and shape of such regions
would be an important factor in determining the validity of
the resulting graph.

Internally, factorOracle uses a linked list of state objects
to store data. The struct for each state $m$ includes:
• the immediate transition to state \( m + 1 \)
• the number of outgoing transitions
• the indices of states terminating the outgoing transitions
• the index of the state terminating the suffix link from state \( m \)

Note that it is not necessary to store both the state indices and the transitions themselves, since a transition from state \( m \) to state \( m + n \) is equal to the transition stored at state \( m + n - 1 \) (Allauzen, Crochemore, and Raffinot 1999). In a subsequent version, each state will also store the longest repeated suffix indicated by its suffix link.

When a bang is sent to the left inlet, transitions are chosen using an internal process selectable by sending a “mode” message with an integer argument, specifying the mode of traversal, to any inlet. There is at present only one such mode, which makes use of a probability value sent as a float to the rightmost inlet. This value can vary in real time. The probability of taking a suffix link \( S \) is related to the probability of taking a transition \( T \) by \( P(S) = 1.0 - P(T) \). Uniform weights are used when choosing among transitions along forward links from the current state, including the link pointing to the next state.

When a suffix link is taken, the mode outputs the transition to the state immediately following the state reached by the suffix link, prohibiting forward skips until the next output request. This avoids one of the problems mentioned in (Assayag and Bloch 2007): if we allow the possibility of forward state skips, we can potentially follow a suffix link and then jump right back to the origin state of the suffix link, producing—in this implementation—unwanted repetitions of a single transition. Unlike Assayag and Bloch, we don’t prohibit forward skips altogether, though perhaps this approach will be taken with a separate generative mode in a future version.

A second issue, noted in both (Assayag and Bloch 2007) and (Dubnov and Wang 2014), involves the meta-rhythmic “errors”—isolated metric idiom violations—that can arise from taking forward leaps or suffix links based on a very short common suffix. These produce jarring “interruptions” or accents, creating a relationship of output to input analogous to the relationship between a Picasso painting and its subject. If these stylistic anomalies are unacceptable to the user, the input word can be built from a cross-alphabet of metric groups of notes, instead of a cross-alphabet of individual notes, or a custom external ruleset designed to avoid incomplete beats can be programmed for use with the middle inlet.

Note that forcing the generative process to take an immediate forward transition after following a suffix link can lead to an infinite loop at the level of a single transition, in the case where the final state has a suffix link back to the previous state. To remedy this, we force the oracle to follow the chain of suffix links back from the terminal state until we reach a state with a suffix link going back more than one state. This is the only case in which we allow the generative state of the oracle to reach zero; in all other cases, we go back no further than state 1, since there is no suffix prior to state zero to exploit as a shared connection between substrings.

### Examples

Figures 4-7 illustrate the variation in output with different probability values \( P \) for the internal generative mode. Figure 3 shows the original head for the tune Donna Lee by Charlie Parker, labelled with cross-alphabet values calculated from the sets of unique pitch \( (p) \) and rhythmic \( (r) \) values shown above the score. For example, middle-C, \( p = 3 \), with a duration of an eighth note, \( r = 1 \), is encoded as \( (r + 25 + p) = 28 \).
If a high degree of stylistic coherence is desired, using a higher value for $P$ will produce longer segments of unrecombined material. Note that Figure 5 is stylistically very similar to the original, but recombined in a novel way, with just a few of the rhythmic anomalies discussed in the previous section.

![Figure 4: Donna Lee with cross-alphabet labels.](image)

![Figure 5: Donna Lee recombined with $P = 0.9$.](image)

The anomalies result from the fact that our input alphabet is comprised of single notes, rather than metric groupings of notes; the 1-beat triplet embellishments in the original are occasionally broken up, causing an interpenetration of duple and triple meters.

This effect is more pronounced in Figures 6 and 7, where incomplete triplet figures produce long chains of metric displacement. Using a lower probability in conjunction with an alphabet that ignores metrical grouping can be interesting if the goal is to distort the input phraseology. Again, this is an idiosyncrasy of the one internal generative mode developed thus far, and can be circumvented. A patch could be made, for example, to use the center inlet for choosing the next transition, returning a transition only if it completes the beat and following a suffix link if no such transition exists.
File I/O

The factorOracle external can write the input string and the generated output string to disk. These files are simple space-separated lists of integers. The oracle can read these lists of integers in two different ways: the second argument in the body of the object icon can be a read-in file in the Max path, or a “read” message can be sent to any inlet, with or without a file argument.

The first argument in the body of the object specifies the number of states anticipated. Reserving memory for a number of states vastly improves performance by avoiding frequent allocating and copying of memory blocks to resize buffers. The default internal number of states is 10K; if the user specifies both a state number limit and an input file, enough memory is allocated for both the input file and the number of states requested in the first argument. Finally, a “writejson” message in any inlet writes the graph to disk in a custom JSON format.

The top-level hash keys shown in Figure 8 are state indices. The value of each of these keys is an array containing a hash of end-state–transition key-value pairs, followed by the index of the state terminating the suffix link. The motivation behind the JSON hash is to provide a standard way of exporting the oracle graph for use in other programming environments, such as Processing.

```json
{
  "0": [{"1": "0", "2": "1"}, "-1"],
  "1": [{"2": "1"}, "0"],
  "2": [{"3": "1"}, "0"],
  "3": [{}, "2"]
}
```

Figure 8: JSON representation for factor oracle of ABB.

Planned Improvements

The most recent version of factorOracle is an alpha phase release, verified to work in Max 7 in 32-bit mode on Mac OS Mavericks, and is available in an open-source BSD-licensed version1. A more stable release will be available soon.

In terms of new development, longest repeated suffix computation is first in the queue. A trickier problem is how to incorporate data from files that are read in after some performance input, or how to incorporate performance input after a file is read in. One idea is to merge the two based on a longest repeated suffix that preserves the most data from both sets of information, which means waiting for or expecting some minimum accumulation of performance data before fully integrating a read-in file. Performance data and file data are presently concatenated without concern for any discontinuity at the point of connection.

Finally, I would like to produce a series of help files showing how to build patches around the external using many of the analytical and generative methods described in the list of references here. One important caveat to note: although the objective of this software is to provide an interface for both constructing the factor oracle and using it to generate music in a real-time audio scheduling environment, the extent to which a patch incorporating the factorOracle external can generate music in real time will be contingent on the computational demands of the generative process employed.

References


