Generation of Gregorian chant melodies through adaptive techniques

P. R. M. Cereda and J. José Neto

***Abstract*—A significant subset of formation rules from the Gregorian theory may be used for an automatic generation of musical compositions. In this paper, we present a set of adaptive techniques for generating Gregorian chant melodies given an in input text in any language, provided that syllabic division rules are available, and both melody mode and variation.**

***Keywords:*—adaptive devices, musical composition, adaptive techniques, syntactic transformations**

I. Introduction

GREGORIAN chant is a form of sung prayer which the Catholic Church has officially adopted for Western Christianity of the Roman Rite [1], [2], [3]. The name is a tribute to Pope Gregory I (also known as Saint Gregory the Great), whose papacy lasted from 540 to 640. Although the chant origins predate greatly this epoch (experts estimate the formation period from the beginning of the Church, in particular, from the end of the persecutions circa 313), Gregory contributed with its development and diffusion [1], [3]. The Pope collected, chose and gave order to the pieces within religious services, and founded the Schola Cantorum, an advanced school for church music [2]. According to Auguste Le Guennant, a French composer and former student, "in Gregorian chant, the text supplies man with the food necessary for his mind; the music provides him with the substance which his heart needs. Thus both contribute to the complete fulfillment of the human being in his relationship with God" [1].

More recently, the formation rules of Gregorian chant became well established mostly due to the enormous efforts of the monks at Saint Peter's Abbey, a Benedictine monastery in Solesmes (Sarthe, France). In particular, a decree of the Sacred Congregation of Rites dated April 11, 1911, authorized Solesmes to edit the Vatican edition with their findings. Since then, the abbey became the reference in Gregorian chant [4], [5].

Given the straightforward structure of a Gregorian chant melody, and inspired by a seminal work by Basseto on using adaptive technology for musical composition [6], [7], we aim at providing adaptive techniques for generating such melodies. A melody is composed based on an input text in any language (provided that syllabic division rules are available) and two parameters, namely mode and variation (detailed later on in Section II). Two files are generated as result of the musical composition: a PDF file with the chant score (properly typeset in Gregorian notation) and a MIDI file with the corresponding melody as a set of instructions to be executed by an electronic device (e.g, a sound card) as a means to produce sound.

Authors can be contacted through the following electronic mail addresses: paulo.cereda@usp.br and jjneto@usp.br.

The remainder of this paper is as follows: Section II presents an overview of the basic concepts used throughout this paper, namely adaptive devices and the Gregorian chant theory. In Section III, we discuss the melody generation itself, addressing mode selection, neume construction, syllabic division and the output format through syntactic transformations. In Section IV, we present an experiment as a means to evaluate the adaptive techniques, including a melody generation from a famous Portuguese poem. In Section V, we list a subset of additional stylistic plainchant guidelines as a means to significantly enhance the final result. Finally, we present our final remarks in Section VI.

II. Preliminary concepts

THIS section formally introduces the concept of a general rule-driven adaptive device, as well as a primer on Gregorian notation and formation rules. It is important to observe that some details on Gregorian chant theory were deliberately omitted as they are not in the scope of our paper.

A. Adaptive devices

A general adaptive rule-driven device consists of an underlying, potentially non-adaptive, rule-driven device enhanced with an extension, namely the adaptive mechanism, that allows modifications on the current rule set over time, based on history and input stimuli [8], [9]. The following definitions offer a formal description on how such device functions from a theoretical perspective.

Definition 1 (rule-driven device)**.** A rule-driven device is defined as $ND = (C, NR, S, c_0, A, NA)$, such that ND is a rule-driven device, C is the set of all possible configurations, $c_0 \in C$ is the initial configuration, S is the set of all possible input stimuli, $\epsilon \in S$, $A \subseteq C$ is the subset of all accepting configurations (respectively, $F = C - A$ is the subset of all rejecting configurations), NA is the set of all possible output stimuli of ND as a side effect of rule applications, $\epsilon \in NA$, and NR is the set of rules defining ND as a relation $NR \subseteq C \times S \times C \times NA$. □

Definition 2 (rule)**.** A rule $r \in NR$ is defined as $r = (c_i, s, c_j, z)$, $c_i, c_j \in C$, $s \in S$ and $z \in NA$, indicating that, as response to a stimulus s, r changes the current configuration c_i to c_j, processes s and generates z as output [8]. A rule $r = (c_i, s, c_j, z)$ is said to be compatible with the current configuration c if and only if $c_i = c$ and s is either empty or equals the current input stimulus; in this case, the application of a rule r moves the device to a configuration c_j, denoted by $c_i \Rightarrow^s c_j$, and adds z to the output stream. □

Definition 3 (acceptance of an input stimuli stream by a rule–driven device)**.** An input stimuli stream $w = w_1 w_2 \dots w_n$, $w_k \in S - \{\epsilon\}$, $k = 1, \dots, n$, $n \geq 0$, is accepted by a device ND when $c_0 \Rightarrow^{w_1} c_1 \Rightarrow^{w_2} \dots \Rightarrow^{w_n} c$ (in short, $c_0 \Rightarrow^{w} c$), and $c \in A$. Respectively, w is rejected by ND when $c \in F$. The language described by a rule-driven device ND is represented by $L(ND) = \{w \in S^* \mid c_0 \Rightarrow^{w} c, c \in A\}$. □

Definition 4 (adaptive rule-driven device)**.** A rule-driven device $AD = (ND_0, AM)$, such that ND_0 is a device and AM is an adaptive mechanism, is said to be adaptive when, for all operation steps $k \geq 0$ (k is the value of an internal counter T starting in zero and incremented by one each time a non-null adaptive action is executed), AD follows the behavior of an underlying device ND_k until the start of an operation step $k + 1$ triggered by a non-null adaptive action, modifying the current rule set; in short, the execution of a non-null adaptive action in an operation step $k \geq 0$ makes the adaptive device AD evolve from an underlying device ND_k to ND_{k+1}. □

Definition 5 (operation of an adaptive device)**.** An adaptive device AD starts its operation in configuration c_0, with the initial format defined as $AD_0 = (C_0, AR_0, S, c_0, A, NA, BA, AA)$. In step k, an input stimulus move AD to the next configuration and starts the operation step $k + 1$ if and only if a non-adaptive rule is executed; thus, being the device AD in step k, with $AD_k = (C_k, AR_k, S, c_k, A, NA, BA, AA)$, the execution of a non-null adaptive action leads to $AD_{k+1} = (C_{k+1}, AR_{k+1}, S, c_{k+1}, A, NA, BA, AA)$, in which $AD = (ND_0, AM)$ is an adaptive device with a starting underlying device ND_0 and an adaptive mechanism AM, ND_k is an underlying device of AD in an operation step k, NR_k is the set of non-adaptive rules of ND_k, C_k is the set of all possible configurations for ND in an operation step k, $c_k \in C_k$ is the starting configuration in an operation step k, S is the set of all possible input stimuli of AD, $A \subseteq C$ is the subset of accepting configurations (respectively, $F = C - A$ is the subset of rejecting configurations), BA and AA are sets of adaptive actions (both containing the null action, $\epsilon \in BA \cap AA$), NA, with $\epsilon \in NA$, is the set of all output stimuli of AD as side effect of rule applications, AR_k is the set of adaptive rules defined as a relation $AR_k \subseteq BA \times C \times S \times C \times NA \times AA$, with AR_0 defining the starting behavior of AD, AR is the set of all possible adaptive rules for AD, NR is the set of all possible underlying non-adaptive rules of AD, and AM is an adaptive mechanism, $AM \subseteq BA \times NR \times AA$, to be applied in an operation step k for each rule in $NR_k \subseteq NR$. □

Definition 6 (adaptive rules)**.** Adaptive rules $ar \in AR_k$ are defined as $ar = (ba, c_i, s, c_j, z, aa)$ indicating that, as response to an input stimulus $s \in S$, ar initially executes the prior adaptive action $ba \in BA$; the execution of ba is canceled if this action removes ar from AR_k; otherwise, the underlying non-adaptive rule $nr = (c_i, s, c_j, z)$, $nr \in NR_k$ is applied and, finally, the post adaptive action $aa \in AA$ is applied [8]. □

Definition 7 (adaptive function)**.** Adaptive actions may be defined as abstractions named adaptive functions, similar to function calls in programming languages [8]. The specification of an adaptive function must include the following elements: *(a)* a symbolic name, *(b)* formal parameters which will refer to values supplied as arguments, *(c)* variables which will hold values of applications of elementary adaptive actions of inspection, *(d)* generators that refer to new value references on each usage, and *(e)* the body of the function itself. □

Definition 8 (elementary adaptive actions)**.** Three types of elementary adaptive actions are defined in order to perform tests on the rule set or modify existing rules, namely:

- inspection: the elementary action does not modify the current rule set, but allows inspection on such set and querying rules that match a certain pattern. It employs the form $?\langle$pattern$\rangle$.
- removal: the elementary action removes rules that match a certain pattern from the current rule set. It employs the form $-\langle$pattern$\rangle$. If no rule matches the pattern, nothing is done.
- insertion: the elementary action adds a rule that match a certain pattern to the rule set. It employs the form $+\langle$pattern$\rangle$. If the rule already exists in the rule set, nothing is done.

Such elementary adaptive actions may be used in the body of an adaptive function, including rule patterns that use formal parameters, variables and generators available in the function scope. □

According to Cereda and José Neto [10], [11], [12], adaptive devices offer conveniences on a more compact model representation and better organization, as each underlying device covers a specific context at a time. Evolutions reflect device fitting based on history and input stimuli in order to accommodate covered yet unexpected scenarios [13], [14], [15], [16], [17].

B. Gregorian notation and formation rules

Historically, the Gregorian notation started as neumatic (also known as chironomic), stemming mainly from the acute and grave accents borrowed from Latin grammar [5], [1]. It was very imperfect due to the absence of a proper staff and thus impossible to indicate the intervals which the voice was to sing. However, it was a very rich notation in indications of the expressive interpretation [3].

The notation evolved to an alphabetic representation, borrowed from the Greeks, in which notes were indicated by letters (from A to G, similar to chord names used in modern popular music) [3]. Although precise in regard to the intervals, the notation was inadequate as to the unity of the neumes [1].

Further enhancements included the indication of intervals (diastematic notation), using the lines which were gradually increased to the number of four, which today forms the Gregorian staff [1]. Neumes were transferred to the staff, such that the primitive accents became points which could be located with precision on the staff. However, at the expense of intervallic precision, the rhythmic details disappeared [2], [3].

As time went by, schools modified the graphic forms by including complementary signs or letters to them, as a means

to determine the length, the brevity or the expressiveness of certain groups or notes. Eventually, the Gregorian experts (in particular at Saint Peter's Abbey, in Solesmes) arrived at a stabilization of the traditional rhythmic interpretation [1].

The basics of Gregorian notation consist of notes arranged on a staff of four lines [18], [19]. The name of the notes is determined by two kinds of clef (C or F), and all notes are equal in length, unless a supplementary sign is interposed. A neume is the graphical sign that represents one or more notes, as well as intervallic and rhythmic indications. The notation has two fundamental neumes: virga and punctum. The movement is regular (also known as isochronic), provided that no supplementary sign is identified (namely, the horizontal episema and the mora dot). Figure 1 outlines the basic elements (clefs, fundamental neumes and staff) [19], [1].

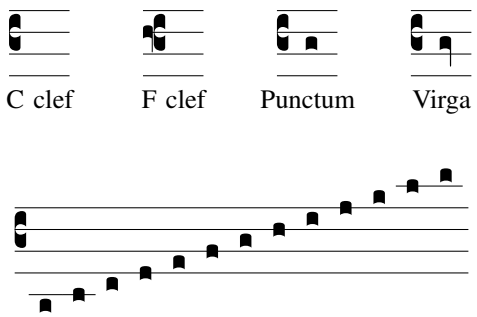

Figure 1. Basic elements of Gregorian notation: clefs (C and F), fundamental neumes (punctum and virga) and the staff itself (composed of four lines plus two additional lines to accommodate more notes). The position of the clef in the staff determines the name of the note.

The punctum and the virga constitute the basis of neume construction in Gregorian chant. The following list presents a non-exhaustive description of neumatic combinations:

- For two different sounds, we have podatus and clivis. The former is an ascending neume composed of a punctum and a virga, while the latter is quite the opposite, i.e, a descending neume composed of a virga and a punctum [18], [1], [2].
- For two identical sounds, we have bivirga, which is made of two virga combined, and distropha, which combines two punctum [18], [19].
- For three different sounds, we have torculus (composed of a punctum, a virga, and a punctum), porrectus (the opposite of torculus, i.e, composed of a virga, a punctum, and a virga), trivirga (as the name indicates, the neume is composed of three virga), and tristopha (three punctum combined) [19], [5].
- For three sounds or more in the same melodic direction, we have climacus (descending neume made up of one virga followed by two, three or four diamond shaped punctum), a scandicus (ascending neumes), and a salicus (ascending neumes, such that two final notes form a podatus and the middle note has a vertical episema) [5], [18].

Observe that certain neumes might end with note heads significantly smaller than those normally used; such neumes are named liquescent. The liquescent note or notes lose part of their clarity and force, but not their length [2], [1].

There are three developed neumes, namely resupinus, flexus and subpunctis. A resupinus neume moves towards a downward movement and rise again on one extra note in an upward direction. A flexus neume acts the opposite, i.e, the neume moves towards an upward movement and is deflected backward in a downward movement. Finally, a subpunctis neume moves towards an upward movement and follows a line of descending diamond-shaped punctum [1], [2].

At last, there are two special neumes: quilisma and oriscus. The former calls for the expressive lengthening of the note or two notes which precede it. The latter is an apostropha added to the last note of a neume [18], [19]. Figure 2 presents the visual representation of neumatic combinations on a Gregorian staff.

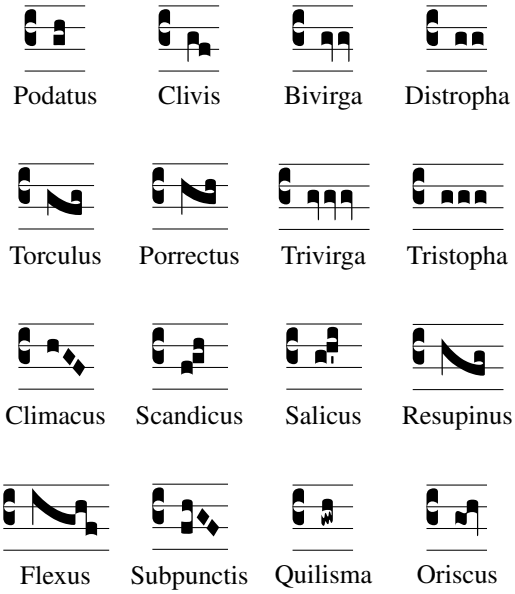

Figure 2. Visual representation of neumatic combinations.

Regarding staff elements, bars are not of measure, but signs of musical punctuation. Also, the flat (♭) is the only chromatic alteration available and affects exclusively the note B [19], [2].

The Gregorian melody is monodic (no polyphonic and harmonic elements and combinations), diatonic and modal (constructed on melodic scales which differ widely from another), besides of having no leading tone [1], [19]. Four modes are available and may be distinguished from inspecting the lowest note, namely protus (starts from D), deuterus (starts from E), tritus (starts from F), and tetrardus (starts from G). There are also two subdivisions on each mode, namely authentic and plagal. Each mode dictates a different message to the hearer [18], [1], [4].

III. Melody generation

CONSIDER a Unicode text written in any language (provided that syllabic division rules are available) and both Gregorian mode and variation (authentic or plagal) as input of our generator. The following subsections provide details on how a melody is generated through adaptive techniques.

A. Mode selection

Notes in the Gregorian staff will be represented by positive integers, from 1 to 13. This approach aims at maintaining an order relation (which is transitive and anti-symmetric) such that, given two notes a and b, $a \geq b$ means that a is possibly in a higher position than b in the staff (hence a higher pitch). Figure 3 presents an annotated Gregorian staff.

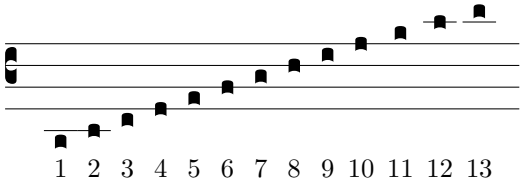

Figure 3. Annotated Gregorian staff, in which notes are represented by positive integers. Given the clef in the third line, the note C is found in indices 1 and 8 in the staff.

The annotated Gregorian staff from Figure 3 is represented by an adaptive automaton N, introduced in Figure 4. Observe that states from 1 to 13 represent the corresponding notes in the staff, while the initial state indicates that no note was selected so far. At first, all notes are unreachable from the initial state; the adaptive function $\mathcal{A}$ (Algorithm 1) is responsible for adding transitions to valid notes based on the selected mode.

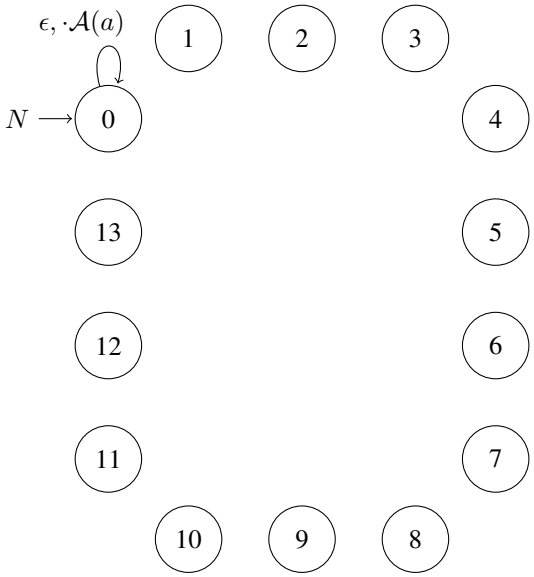

Figure 4. Adaptive automaton N representing notes from the annotated Gregorian staff. Adaptive function $\mathcal{A}$ is presented in Algorithm 1.

Algorithm 1 Adaptive function $\mathcal{A}$

adaptive function $\mathcal{A}(a)$
$-(0, \epsilon) \rightarrow 0, \cdot\mathcal{A}(a)$
$+(0, \epsilon) \rightarrow \{i, \cdot\psi_{rand}(i) \mid a \leq i \leq (a + 7)\}$
end of adaptive function

Observe that $\mathcal{A}$ (Algorithm 1) is a parametric function. A Gregorian mode selects a range of notes from the staff, so this particular adaptive function restricts which states are reachable from the start by taking a mode shift as parameter. Based on the existing modes and variations briefly described in Subsection II-B, let there be two sets M and V, such that $M = \{\text{protus}, \text{deuterus}, \text{tritus}, \text{tetrardus}\}$ and $V = \{\text{authentic}, \text{plagal}\}$. Possible combinations are given by $M \times V$; these entries are presented in Table I, including their corresponding shifts.

Table I
SHIFTS BASED ON THE COMBINATION OF MODE AND VARIATION FROM GREGORIAN MELODY DESCRIBED IN SUBSECTION II-B.

	Variations	
Modes	Authentic	Plagal
Protus	2	6
Deuterus	3	1
Tritus	4	1
Tetrardus	5	2

For instance, consider an authentic protus mode for a Gregorian melody (with shift 2 according to Table I). The new configuration of the adaptive automaton N (Figure 4) after the execution of $\mathcal{A}(a)$, with $a = 2$, is presented in Figure 5. Observe that only the protus note range is reachable from the initial state.

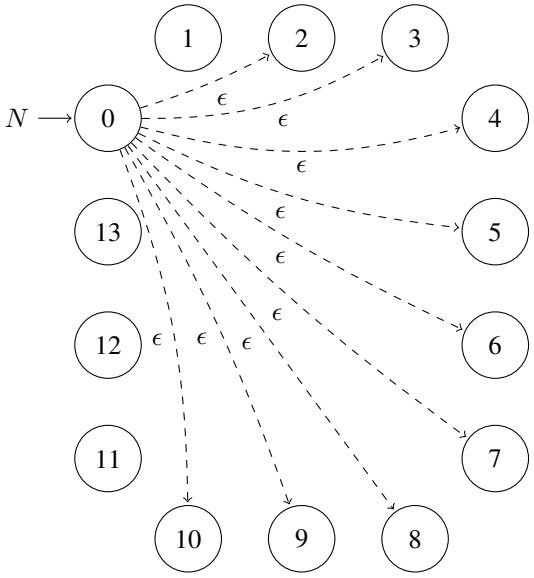

Figure 5. New configuration of the adaptive automaton N (Figure 4) after the execution of $\mathcal{A}(a)$, with $a = 2$, representing the authentic protus mode. Transitions have omitted the associated adaptive function ψ_{rand} due to space constraints.

The newly added transitions (as seen in Figure 5) are represented by dashed lines for a reason: they are bound to probabilities following a uniform distribution. Each note from the mode range is equally eligible for neume construction. As the Gregorian melody has no leading tone, notes share the same probability.

B. Neume construction

Once the adaptive automaton N is shaped to reflect the selected mode, notes are eligible for neume construction.

Observe that, for each new transition added by $\mathcal{A}(a)$, there is an adaptive function $\psi_{rand}(i)$ (with i being a reference to the target state) associated to it. As the subscript indicates, ψ_{rand} is actually a reference to another adaptive function selected at random, such that $\psi_{rand} = \psi \mid \psi \in \{$*punctum/virga*, *podatus*, *clivis*, *scandicus/salicus*, *climacus*, *torculus*, *porrectus*$\}$. These adaptive functions effectively construct the neumes based on the formation rules described in Subsection II-B and use the chosen note as pivot. Table II indicates the adaptive functions available for neume construction and their corresponding algorithms.

Table II
ADAPTIVE FUNCTIONS AVAILABLE FOR NEUME CONSTRUCTION AND THEIR CORRESPONDING ALGORITHMS.

Adaptive function	Algorithm
punctum/virga	2
podatus	3
clivis	4
scandicus/salicus	5
climacus	6
torculus	7
porrectus	8

Algorithm 2 Adaptive function *punctum/virga*

adaptive function *punctum/virga*(a)
$-(\{i \mid 1 \leq i \leq 13\}, \epsilon) \rightarrow \{i \mid 1 \leq i \leq 13\}$
end of adaptive function

Algorithm 3 Adaptive function *podatus*

adaptive function *podatus*(a)
variables: $?x$
$-(\{i \mid 1 \leq i \leq 13\}, \epsilon) \rightarrow \{i \mid 1 \leq i \leq 13\}$
$?(0, \epsilon) \rightarrow ?x$
$+(a, \epsilon) \rightarrow i \mid i \in ?x, i > a$
end of adaptive function

Algorithm 4 Adaptive function *clivis*

adaptive function *clivis*(a)
$-(\{i \mid 1 \leq i \leq 13\}, \epsilon) \rightarrow \{i \mid 1 \leq i \leq 13\}$
$?(0, \epsilon) \rightarrow ?x$
$+(a, \epsilon) \rightarrow i \mid i \in ?x, i < a$
end of adaptive function

After a closer inspection on Table II, it is clear that not all formation rules from Subsection II-B are covered through adaptive rules. This is a deliberate choice, since the remainder is quite straightforward (e.g, a bivirga is simply two virga combined, with the same note). Also, observe that previous neume construction schemes are cleaned up before a new one takes place (indicated by the first line of each adaptive function).

For instance, consider an example of a climacus construction within an authentic protus mode (Figure 5). Given 9 as the chosen note (namely, D), the execution of the adaptive function *climacus*(9) might yield (amongst several others) the new configuration as seen in Figure 6.

Algorithm 5 Adaptive function *scandicus/salicus*

adaptive function *scandicus/salicus*(a)
$-(\{i \mid 1 \leq i \leq 13\}, \epsilon) \rightarrow \{i \mid 1 \leq i \leq 13\}$
$?(0, \epsilon) \rightarrow ?x$
$w \leftarrow (i, j) \mid (i, j) \in ?x \times ?x, a < i < j$
$+(a, \epsilon) \rightarrow i \mid (i, \alpha) \in w$
$+(i \mid (i, \alpha) \in w, \epsilon) \rightarrow i \mid (\alpha, i) \in w$
end of adaptive function

Algorithm 6 Adaptive function *climacus*

adaptive function *climacus*(a)
$-(\{i \mid 1 \leq i \leq 13\}, \epsilon) \rightarrow \{i \mid 1 \leq i \leq 13\}$
$?(0, \epsilon) \rightarrow ?x$
$w \leftarrow (i, j) \mid (i, j) \in ?x \times ?x, i < j < a$
$+(a, \epsilon) \rightarrow i \mid (\alpha, i) \in w$
$+(i \mid (i, \alpha) \in w, \epsilon) \rightarrow i \mid (i, \alpha) \in w$
end of adaptive function

According to Figure 6, the generated climatus neume is composed of notes 9, 7 and 4 (D, B and F, respectively), in that order, for an authentic protus mode. Other combinations of notes are possible. As the neume construction process is fully covered, it suffices to inspect the provided text in order to discover how many neumes are needed to be constructed, so the entire phrasal structure is properly addressed.

C. Syllabic division

Given the Unicode text provided as input, we need to break it into parts. The first step is to split the text using empty spaces as separators and thus obtain a list of words. From this list, every word is then divided into syllables. Figure 7 outlines the text manipulation process. Note that the module requires a proper language definition containing the syllabic division

Algorithm 7 Adaptive function *torculus*

adaptive function *torculus*(a)
$-(\{i \mid 1 \leq i \leq 13\}, \epsilon) \rightarrow \{i \mid 1 \leq i \leq 13\}$
$?(0, \epsilon) \rightarrow ?x$
$w \leftarrow i \mid i \in ?x, i > a$
$+(a, \epsilon) \rightarrow w$
$+(w, \epsilon) \rightarrow a$
end of adaptive function

Algorithm 8 Adaptive function *porrectus*

adaptive function *porrectus*(a)
$-(\{i \mid 1 \leq i \leq 13\}, \epsilon) \rightarrow \{i \mid 1 \leq i \leq 13\}$
$?(0, \epsilon) \rightarrow ?x$
$w \leftarrow i \mid i \in ?x, i > a$
$y \leftarrow i \mid i \in ?x, i < a$
$+(a, \epsilon) \rightarrow w$
$+(w, \epsilon) \rightarrow y$
end of adaptive function

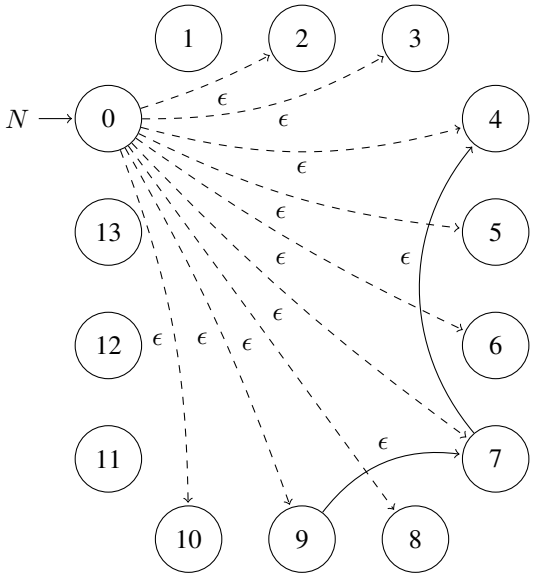

Figure 6. A possible new configuration after the execution of the adaptive function *climacus*(9) within an authentic protus mode (Figure 5).

rules (based on Hunspell, a spell checker and morphological analyzer designed for languages with rich morphology and complex word compounding and character encoding) in order to properly process each word.

For instance, consider the syllabic division of the word `constitutional`, as seen in Figure 8. The module requires a word and the associated language definition file (in this case, English). As a result, five syllables are produced.

The total of neumes is calculated upon the sum of all syllables of all words from the list. Formally, let there be $D\colon \Sigma^* \mapsto \mathbb{Z}$ a function that maps words into their number of syllables, and L the list of all words extracted from the input text, the total of neumes TN is presented in Equation 1.

$$TN = \sum_{l \in L}^{|L|} C(l) \tag{1}$$

In order to accommodate all syllables from all words in the text, we need to generate TN neumes using the construction scheme detailed in Subsection III-B. Then each neume is associated with a syllable and included in a syntactic structure (namely a tree) for later processing.

D. Output

Neumes and their associated syllables are stored in a tree structure, as it gives an expressive representation of all elements. Figure 9 presents an example of a tree containing four neumes and the corresponding syllables for the word `irregular` (using English syllabic division rules).

In order to generate the output files (PDF and MIDI), we will apply syntactic transformations on this tree and shape the elements according to each format. Figure 10 outlines the transformation process.

Two independent syntactic transformations are applied to the tree of neumes and syllables in order to obtain the output files seen in Figure 10. They are described as follows:

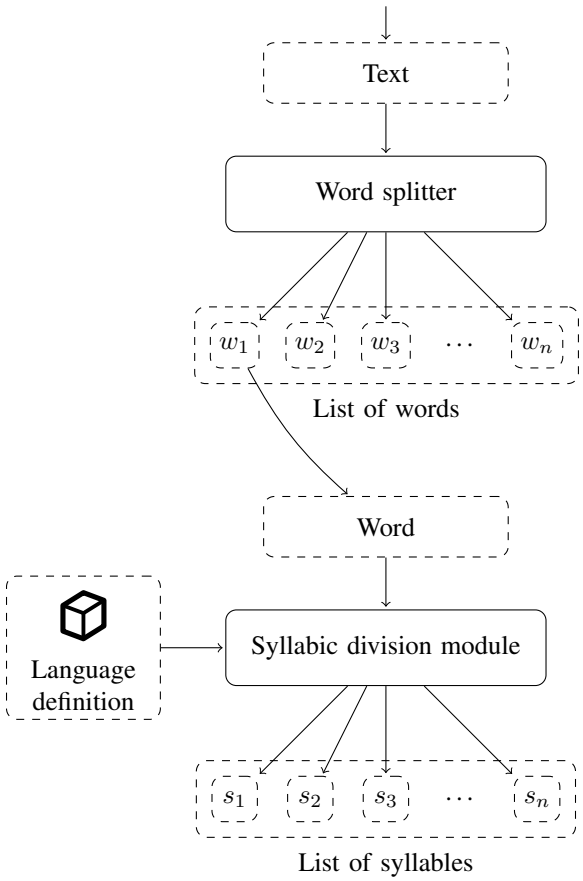

Figure 7. Text manipulation process overview. Note that the module requires a proper language definition containing the syllabic division rules in order to properly process each word.

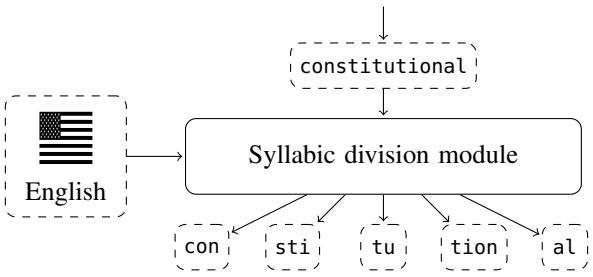

Figure 8. Syllabic division module.

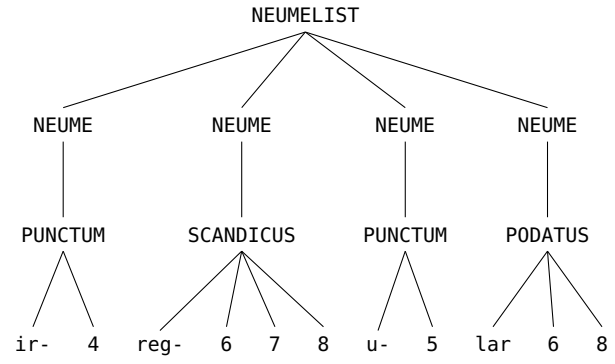

Figure 9. Example of a tree structure containing four neumes and the corresponding syllables for the English word `irregular`. Values are represented in the leaves.

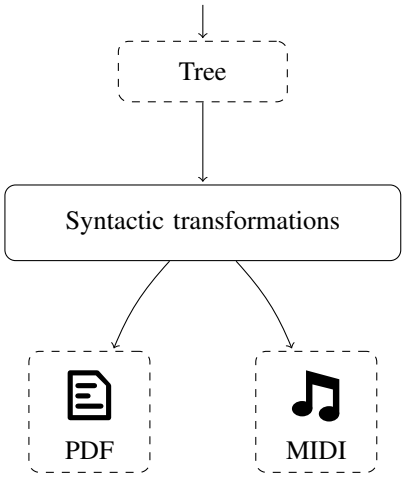

Figure 10. Outline of syntactic transformations applied to the tree, resulting in both PDF and MIDI output formats.

Os Lusíadas
Luís de Camões

Canto I

As armas e os barões assinalados,
Que da ocidental praia Lusitana,
Por mares nunca de antes navegados,
Passaram ainda além da Taprobana,
Em perigos e guerras esforçados,
Mais do que prometia a força humana,
E entre gente remota edificaram
Novo Reino, que tanto sublimaram.

Figure 11. *Canto I* from the Portuguese poem *Os Lusíadas*, by Luís Vaz de Camões, to be used as input text of our experiment.

- The PDF transformation takes the tree and reduces it to an intermediate format known as GABC, a simple text-based notation that enables the description of Gregorian chant scores. Once the transformation is done, the file is processed by an external program and a PDF file containing the typeset chant melody score is generated.
- The MIDI transformation takes the tree, removes all textual references (namely, syllables) and reduces it to an intermediate format known as LY, another simple text-based notation for music input. Once the transformation is done, we deliberately suppress the score engraving and explicitly declare a MIDI generation. Then the file is processed by an external program and a MIDI file with the corresponding chant melody as a set of instructions to be executed by an electronic device is generated.

After the generation of both PDF and MIDI files, the intermediate files are removed since they are no longer necessary. Optionally, the generated tree may remain available afterwards for subsequent use and application of external syntactic transformations (through library mode and API calls).

IV. Experiments

In order to evaluate the adaptive techniques presented here, we wrote a study case of a Gregorian chant generation. For this particular experiment, we chose *Canto I* (Figure 11) from the Portuguese poem *Os Lusíadas*, by Luís Vaz de Camões [20]. The poem is often regarded as the most important work of Portuguese literature.

Consider the program interface illustrated in Figure 12. The program presents us with a straightforward interface in which we simply type the input text and select both mode and text language. Observe that the program automatically detected installed Hunspell dictionaries and listed them (using IETF language tag names) in the selection box.

We typed the contents of Canto I (as seen in Figure 11), as a single paragraph, in the input area, selected the authentic protus mode (displayed as an order pair $(m, v) \in M \times V$, previously defined in Subsection III-A) and the European Portuguese language definition (identified as the `pt_PT` tag), clicked the "Generate chant" button and then waited a few moments while the neume construction, syllabic division and syntactic transformations took place. The resulting chant melody score is presented in Figure 13.

As result of the melody generation, both PDF and MIDI files were successfully obtained from the program interface. Observe that the provided text was successfully broken into words and then into syllabic elements, which were associated to neumes and inserted into a tree structure. Then syntactic transformations were applied to the tree, in which the expected output files derived.

V. Additional enhancements

The current rule set might be extended in order to include additional stylistic plainchant guidelines as a means to significantly enhance the final result. We list a subset of such guidelines as follows.

- Large intervals between adjacent notes should not be frequent, unless there is a new phrase to be composed. The final repose and cadence of the melody should be established on a sound selected as and orientation point at a lower end of a partial scale of sounds [1].
- Breaks are tied to punctuation symbols such that the rhythm flows according to the sentence and not vice versa. In Gregorian chant, bars are not for measure; they are signs of musical punctuation, in which certain breaks call for breathing, to a greater or lesser extent according to the case at hand [2]:
  - Quarter bar: generally indicates the end of an incise and determines a slight respiration.
  - Half bar: indicates the end of a member and calls for and obligatory breath.
  - Full bar: indicates the end of of a phrase which is expressed by a slight broadening of the movement.
- Sentences are broken into three phrasal structures (namely beginning, middle and end) such that the melody

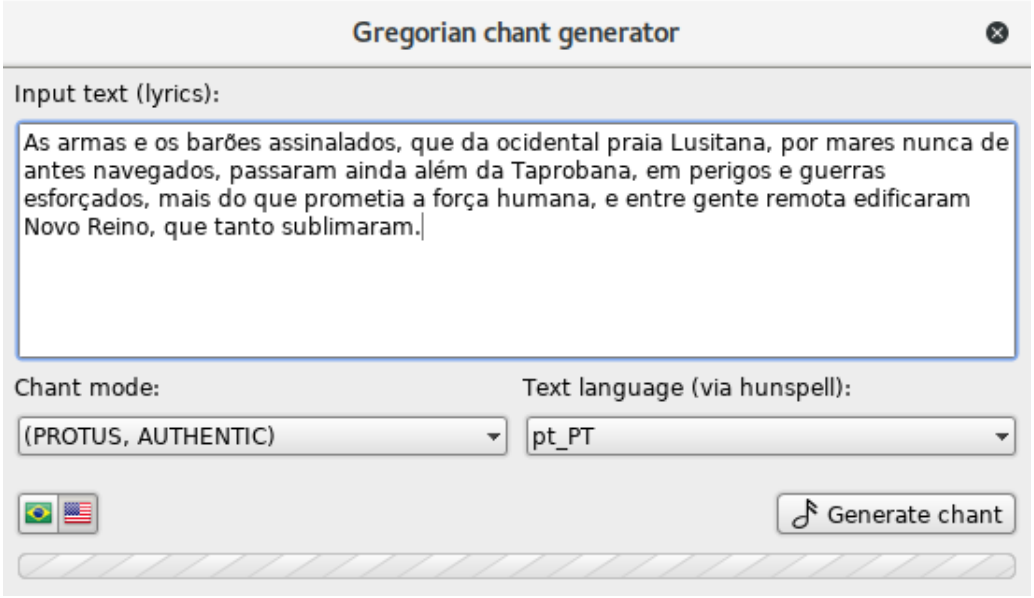

Figure 12. Gregorian chant melody generator interface. It is a simple interface in which we simply type the input text and select both mode and text language.

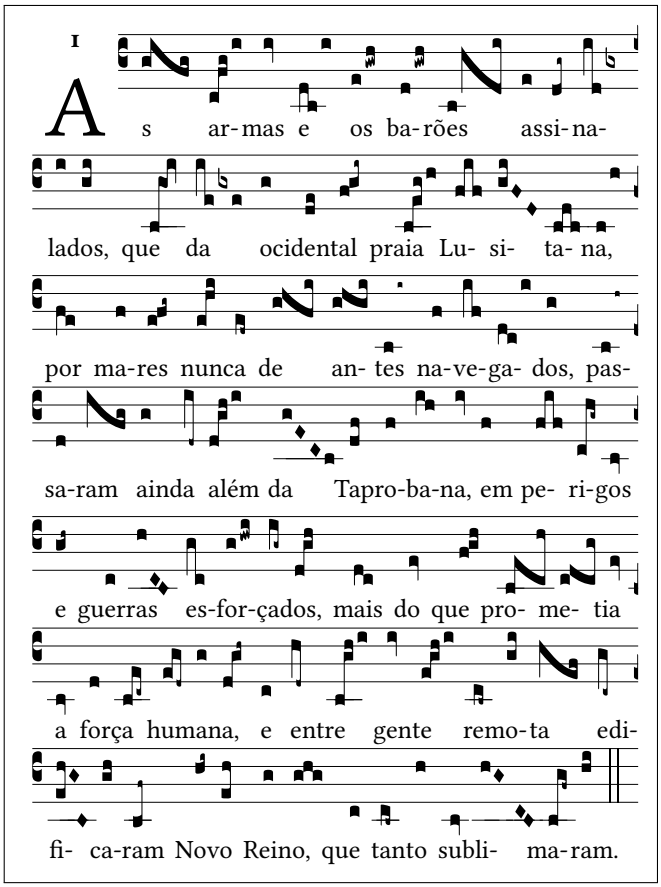

Figure 13. Resulting chant melody score from Canto I (Figure 11).

starts from *thesis* (a Greek word for *repose*), ascends into *arsis* (a Greek word for *impulse*) and returns to its initial state. The movement is neither the arsis or thesis alone, but the two in dependence upon each other, such that the thesis is the natural result of the arsis [18], [1].

- Regarding the underlying language, the tonic syllables are usually emphasized. However, the tonic emphasis can weakened or eliminated based on the surrounding phrasal structure. It is important to note that each language has its own pronunciation rules (e.g, the tonic accent of Latin polysyllabic words is never on the final syllable) [1].

From a stylistic point of view, these guidelines aim at providing a more pleasant experience when generating Gregorian chant melodies, such that the final result is comparable to real compositions in both melodic and rhythmic aspects.

VI. Final Remarks

This paper presented a set of adaptive techniques for generating Gregorian chant melodies given an input text in any language (provided that syllabic division rules are available) and the melody mode and variation. We were able to map a meaningful subset of formation rules from the Gregorian theory into adaptive functions and the results were preliminary yet quite significant.

However, certain context dependencies were minimized for the sake of clarity and simplicity. The Gregorian theory is in itself extensive and intricate. Further studies are needed in order to improve the representation of formation rules and dependencies in both musical and textual elements.

The use of adaptive techniques provides great reduction on the rule set, as it retains only the relevant elements according to the current context. Such feature confers a compact model representation, as context changes are applied only when needed. Additionally, such techniques may be extended towards other genres in musical composition.

REFERENCES

[1] J. R. Carroll, *An applied course in Gregorian chant*, ser. The Church Musicians Bookshelf. Ohio, USA: Gregorian Institute of America, 1956, no. 2.

[2] D. Johner, *A new school of Gregorian chant*, 3rd ed. Frederick Pustet & Co., 1925.

[3] D. Hiley, *Western plainchant: a handbook*. Oxford, UK: Claredon Press, 1993.

[4] D. Saulnier, *Gregorian chant: a guide*. Solesmes, France: Abbaye Saint-Pierre, 2003.

[5] J. Schrembs, A. Marie, and G. Huegle, *The Gregorian Chant Manual of the Catholic Music Hour*. Silver, Burdett and Company.

[6] B. A. Basseto and J. José Neto, "A stochastic musical composer based on adaptive algorithms," in *Proceedings of the VI Brazilian Symposium on Computer Music*, Rio de Janeiro, Brazil, 1999.

[7] B. A. Basseto, "Um sistema de composição musical automatizada, baseado em gramáticas sensíveis ao contexto, implementado com formalismos adaptativos," Tese de mestrado, Escola Politécnica, Universidade de São Paulo, São Paulo, Brazil, 2000.

[8] J. José Neto, "Adaptive rule-driven devices: general formulation and case study," in *International Conference on Implementation and Application of Automata*, 2001.

[9] J. José Neto, "Um levantamento da evolução da adaptatividade e da tecnologia adaptativa," *IEEE Latin America Transactions*, vol. 5, pp. 496–505, 2007.

[10] P. R. M. Cereda and J. José Neto, "AA4J: uma biblioteca para implementação de autômatos adaptativos," in *Memórias do X Workshop de Tecnologia Adaptativa – WTA 2016*, 2016, pp. 16–26.

[11] ——, "Towards performance-focused implementations of adaptive devices," *Procedia Computer Science*, vol. 109, pp. 1164–1169, 2017.

[12] ——, "A middleware architecture for adaptive devices," *Procedia Computer Science*, vol. 109, pp. 1158–1163, 2017.

[13] J. José Neto, "Adaptive automata for context-sensitive languages," *SIGPLAN Notices*, vol. 29, no. 9, pp. 115–124, set 1994.

[14] ——, "Contribuições à metodologia de construção de compiladores," Tese de livre docência, Escola Politécnica da Universidade de São Paulo, São Paulo, 1993.

[15] P. R. M. Cereda and J. José Neto, "Utilizando linguagens de programação orientadas a objetos para codificar programas adaptativos," in *Memórias do IX Workshop de Tecnologia Adaptativa – WTA 2015*, 2015, pp. 2–9.

[16] R. L. Stange, P. R. M. Cereda, and J. José Neto, "Agentes adaptativos reativos: formalização e estudo de caso," in *Memórias do XI Workshop de Tecnologia Adaptativa – WTA 2017*, São Paulo, 2017, pp. 63–71.

[17] R. L. Stange and J. José Neto, "Learning decision rules using adaptive technologies: a hybrid approach based on sequential covering," *Procedia Computer Science*, vol. 109, pp. 1188–1193, 2017.

[18] M. Demetria, *Basic Gregorian chant and sight reading: movable Do edition*, ser. The Church Musicians Bookshelf. Ohio, USA: Gregorian Institute of America, 1960, no. 4.

[19] C. Spence, *Chants of the Church: selected Gregorian chants*. Ohio, USA: Gregorian Institute of America, 1952, edited and compiled by the Monks of Solesmes.

[20] L. V. Camões, *Os Lusíadas*. Companhia Editora do Minho, 1940, edição artística comemorativa do terceiro centenário da restauração da independência de Portugal.

Paulo Roberto Massa Cereda é graduado em Ciência da Computação pelo Centro Universitário Central Paulista (2005) e mestre em Ciência da Computação pela Universidade Federal de São Carlos (2008). Atualmente, é doutorando do Programa de Pós-Graduação em Engenharia de Computação do Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo, atuando como aluno pesquisador no Laboratório de Linguagens e Técnicas Adaptativas do PCS. Tem experiência na área de Ciência da Computação, com ênfase em Teoria da Computação, atuando principalmente nos seguintes temas: tecnologia adaptativa, autômatos adaptativos, dispositivos adaptativos, linguagens de programação e construção de compiladores.

João José Neto é graduado em Engenharia de Eletricidade (1971), mestre em Engenharia Elétrica (1975), doutor em Engenharia Elétrica (1980) e livre-docente (1993) pela Escola Politécnica da Universidade de São Paulo. Atualmente, é professor associado da Escola Politécnica da Universidade de São Paulo e coordena o LTA – Laboratório de Linguagens e Técnicas Adaptativas do PCS – Departamento de Engenharia de Computação e Sistemas Digitais da EPUSP. Tem experiência na área de Ciência da Computação, com ênfase nos Fundamentos da Engenharia da Computação, atuando principalmente nos seguintes temas: dispositivos adaptativos, tecnologia adaptativa, autômatos adaptativos, e em suas aplicações à Engenharia de Computação, particularmente em sistemas de tomada de decisão adaptativa, análise e processamento de linguagens naturais, construção de compiladores, robótica, ensino assistido por computador, modelagem de sistemas inteligentes, processos de aprendizagem automática e inferências baseadas em tecnologia adaptativa.