Memórias do WTA 2018

XII Workshop de Tecnologia Adaptativa

Laboratório de Linguagens e Técnicas Adaptativas
Departamento de Engenharia de Computação e Sistemas Digitais
Escola Politécnica da Universidade de São Paulo

São Paulo
2018

Ficha catalográfica

Workshop de Tecnologia Adaptativa (12: 2018: São Paulo)
Memórias do WTA 2018. – São Paulo; EPUSP, 2018. 58p.

ISBN 978-85-86686-98-6

1. Engenharia de computação (Congressos) 2. Teoria da computação (Congressos) 3. Teoria dos autômatos (Congressos) 4. Semântica de programação (Congressos) 5. Linguagens formais (Congressos) I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II. t.

CDD 621.39

Apresentação

A décima segunda edição do Workshop de Tecnologia Adaptativa realizou-se em São Paulo, Brasil, nos dias 1 e 2 de Fevereiro de 2018, nas dependências da Escola Politécnica da Universidade de São Paulo. As contribuições encaminhadas na forma de artigos relacionados à Tecnologia Adaptativa, nas seguintes áreas, abrangeram, de forma não exclusiva, os tópicos abaixo:

Fundamentos da Adaptatividade

- Modelos de computação, autômatos, gramáticas, grafos e outros dispositivos automodificáveis, suas notações, sua formalização, complexidade, propriedades e comparações com formalismos clássicos.

Tecnologia Adaptativa – Técnicas, Métodos e Ferramentas

- Aplicação dos conhecimentos científicos relativos à adaptatividade e dos dispositivos adaptativos como fundamento para a formulação e para a resolução de problemas práticos.
- Ferramentas, técnicas e métodos para a automatização da resolução de problemas práticos usando técnicas adaptativas.
- Programação adaptativa: linguagens, compiladores e metodologia para o desenvolvimento, implementação e validação de programas com código adaptativo.
- Meta-modelagem de software adaptativo.
- Engenharia de Software voltada para a especificação, projeto, implementação e desenvolvimento de programas automodificáveis de qualidade.
- Adaptatividade multinível e outros conceitos introduzidos recentemente: avanços teóricos, novas ideias para aplicações, sugestões de uso prático.
- Linguagens de alto nível para a codificação de programas automodificáveis: aplicações experimentais e profissionais, práticas e extensas, das novas ideias de uso de linguagens adequadas para a codificação de programas adaptativos e suas metodologias de desenvolvimento.

Aplicações da Adaptatividade e da Tecnologia Adaptativa

- Inteligência computacional: aprendizagem de máquina, representação e manipulação do conhecimento;
- Computação natural, evolutiva e bio-inspirada;
- Sistemas de computação autonômica e reconfigurável;

– Processamento de linguagem natural, sinais e imagens: aquisição, análise, síntese, reconhecimento, conversões e tradução;

– Inferência, reconhecimento e classificação de padrões;

– Modelagem, simulação e otimização de sistemas inteligentes de: tempo real, segurança, controle de processos, tomada de decisão, diagnóstico, robótica;

– Simulação, arte por computador e jogos eletrônicos inteligentes;

– Outras aplicações da Adaptatividade, nas diversas áreas do conhecimento: ciências exatas, biológicas e humanas.

Comissão de Programa

– Almir Rogério Camolesi (Assis, SP, Brasil)

– Ana Cristina Fricke Matte (Belo Horizonte, MG, Brasil)

– André Riyuiti Hirakawa (São Paulo, SP, Brasil)

– Angela Hum Tchemra (São Paulo, SP, Brasil)

– Aparecido Valdemir de Freitas (São Paulo, SP, Brasil)

– Ariane Machado Lima (São Paulo, SP, Brasil)

– Carlos Eduardo Cugnasca (São Paulo, SP, Brasil)

– Claudia Maria Del Pilar Zapata (Lima, Peru)

– Djalma Padovani (São Paulo, SP, Brasil)

– Elisângela Silva da Cunha Rodrigues (Ponta Porã, MS, Brasil)

– Fabrício Augusto Rodrigues (Ponta Porã, MS, Brasil)

– Fátima Nunes (São Paulo, SP, Brasil)

– Francisco Supino Marcondes (São Paulo, SP, Brasil)

– Haroldo Guibu (São Paulo, SP, Brasil)

– Hemerson Pistori (Campo Grande, MS, Brasil)

– Ítalo Santiago Vega (São Paulo, SP, Brasil)

– João Eduardo Kögler Junior (São Paulo, SP, Brasil)

– Leoncio Claro de Barros Neto (São Paulo, SP, Brasil)

– Márcia Ito (São Paulo, SP, Brasil)

– Marcos Pereira Barretto (São Paulo, SP, Brasil)

– Marcus Vinícius Midena Ramos (Petrolina, PE, Brasil)

– Maria Júlia Pascali (Campinas, SP, Brasil)

- Newton Kiyotaka Miura (São Paulo, SP, Brasil)
- Reginaldo Inojosa da Silva Filho (Ponta Porã, MS, Brasil)
- Renata Luiza Stange (Guarapuava, PR, Brasil)
- Ricardo Luís de Azevedo da Rocha (São Paulo, SP, Brasil)
- Ricardo Martins (Minho, Portugal)
- Ricardo Nakamura (São Paulo, SP, Brasil)
- Rosalia Edith Caya Carhuanina (São Paulo, SP, Brasil)
- Wagner José Dizeró (Lins, SP, Brasil)

Comissão Organizadora

- André Riyuiti Hirakawa (São Paulo, SP, Brasil)
- João José Neto, Chair (São Paulo, SP, Brasil)
- Newton Kiyotaka Miura (São Paulo, SP, Brasil)
- Paulo Roberto Massa Cereda (São Paulo, SP, Brasil)
- Ricardo Luís de Azevedo da Rocha (São Paulo, SP, Brasil)

Promoção do Evento

- LTA – Laboratório de Linguagens e Técnicas Adaptativas
- PCS – Departamento de Engenharia de Computação e Sistemas Digitais da EPUSP

Apoio

- Escola Politécnica da Universidade de São Paulo (EPUSP)
- FAPESP – Fundação de Amparo à Pesquisa do Estado de São Paulo, processo número 2017/20900-7
- IEEE – Institute of Electrical and Electronics Engineers
- Olos Tecnologia e Sistemas Ltda.
- Pagga Tecnologia de Pagamentos Ltda.
- São Paulo Convention & Visitors Bureau
- SBC – Sociedade Brasileira de Computação
- SPC – Sociedad Peruana de Computación
- Universidade de São Paulo

Memórias do WTA 2018

Esta publicação do Laboratório de Linguagens e Técnicas Adaptativas do Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo é uma coleção de textos produzidos para o WTA 2018, o Décimo Segundo Workshop de Tecnologia Adaptativa, realizado em São Paulo nos dias 1 e 2 de Fevereiro de 2018. A exemplo da edição de 2017, este evento contou com uma forte presença da comunidade de pesquisadores que se dedicam ao estudo e ao desenvolvimento de trabalhos ligados a esse tema em diversas instituições brasileiras e estrangeiras, sendo o material aqui compilado representativo dos avanços alcançados nas mais recentes pesquisas e desenvolvimentos realizados.

Introdução

Com muita satisfação compilamos neste documento estas memórias com os artigos apresentados no WTA 2018 – Décimo Segundo Workshop de Tecnologia Adaptativa, realizado na Escola Politécnica da Universidade de São Paulo nos dias 1 e 2 de Fevereiro de 2018.

Esta edição contou com 8 trabalhos relacionados à área de Tecnologia Adaptativa e aplicações nos mais diversos segmentos. Adicionalmente, foram ministrados três tutoriais, a saber: *Tutorial do Djalma*, por Djalma Padovani (Escola Politécnica, USP), *AI on Azure: Inteligência Artificial e Machine Learning com Microsoft Azure*, por Rober Torres (Microsoft), e *Construindo metaobjetos em Cyan*, por José de Oliveira Guimarães (Departamento de Computação, UFSCar/Sorocaba). O evento registrou uma participação efetiva de uma centena de pesquisadores durante os dois dias, constatando-se o sucesso do mesmo.

Esta publicação

Estas memórias espelham o conteúdo apresentado no WTA 2018. A exemplo do que foi feito no ano anterior, todo o material referente aos trabalhos apresentados no evento estará acessível no portal do WTA 2018, incluindo softwares e os slides das apresentações das palestras. Adicionalmente, o evento foi gravado em vídeo, em sua íntegra, e os filmes serão também disponibilizados aos interessados. Esperamos que, pela qualidade e diversidade de seu conteúdo, esta publicação se mostre útil a todos aqueles que desejam adquirir ou aprofundar ainda mais os seus conhecimentos nos fascinantes domínios da Tecnologia Adaptativa.

Conclusão

A repetição do sucesso das edições anteriores do evento, e o nível de qualidade dos trabalhos apresentados atestam a seriedade do trabalho que vem sendo realizado, e seu impacto junto à comunidade. Somos gratos aos que contribuíram de alguma forma para o brilho do evento, e aproveitamos para estender a todos o convite para participarem da próxima edição, em 2019.

Agradecimentos

Às instituições que apoiaram o WTA 2018, à comissão organizadora, ao pessoal de apoio e a tantos colaboradores voluntários, cujo auxílio propiciou o êxito que tivemos a satisfação de observar. Gostaríamos também de agradecer às seguintes pessoas pelo valioso auxílio para a viabilização das necessidades locais do evento:

Apoio administrativo

- Ákio Nogueira Barbosa
- André Rodrigues Ribeiro
- Leia Sicília

Apoio operacional

- Daniel Costa Ferreira
- Edson de Souza
- Haroldo Issao Guibu
- Newton Kiyotaka Miura
- Nilton Araújo do Carmo
- Renata Luiza Stange
- Rodrigo Kavakama
- Suellen Alves

Divulgação

- Amanda Panteri
- Amanda Rabelo

De modo especial, agradecemos ao Departamento de Engenharia de Computação e Sistemas Digitais e a Escola Politécnica da Universidade de São Paulo pelo inestimável apoio para a realização da décima segunda edição do Workshop de Tecnologia Adaptativa. Também agradecemos ao Departamento de Engenharia de Computação e Sistemas Digitais pela disponibilização do auditório para a realização do evento e à Fundação de Amparo à Pesquisa do Estado de São Paulo pelo apoio financeiro. Por fim, agradecemos aos engenheiros Newton Kiyotaka Miura e Paulo Roberto Massa Cereda pelo valioso auxílio na organização.

São Paulo, 2 de Fevereiro de 2018

João José Neto
Coordenador geral

Processo número 2017/20900-7, Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP)

As opiniões, hipóteses e conclusões ou recomendações expressas neste material são de responsabilidade do(s) autor(es) e não necessariamente refletem a visão da FAPESP.

Sumário

Lista de autores

Parte I

Submissões

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].

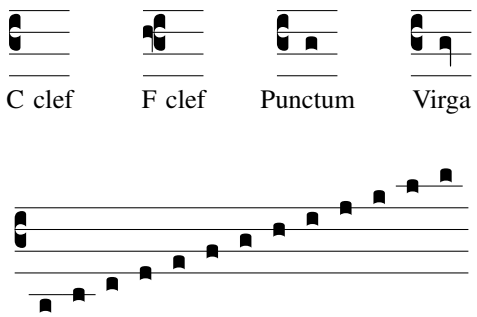


Figure 1. Basic elements of Gregorian notation: clefs (C and F), fundamental neumes (punctum and virga) and the staff itself (composed of four lines plus two additional lines to accommodate more notes). The position of the clef in the staff determines the name of the note.

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.

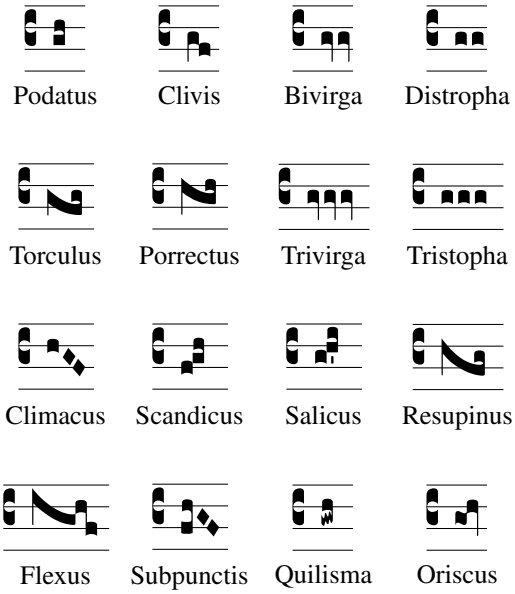


Figure 2. Visual representation of neumatic combinations.

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.

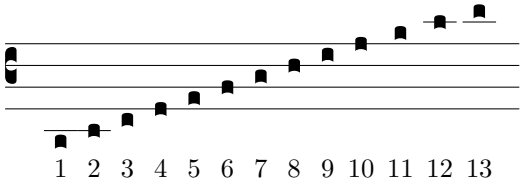


Figure 3. Annotated Gregorian staff, in which notes are represented by positive integers. Given the clef in the third line, the note C is found in indices 1 and 8 in the staff.

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.

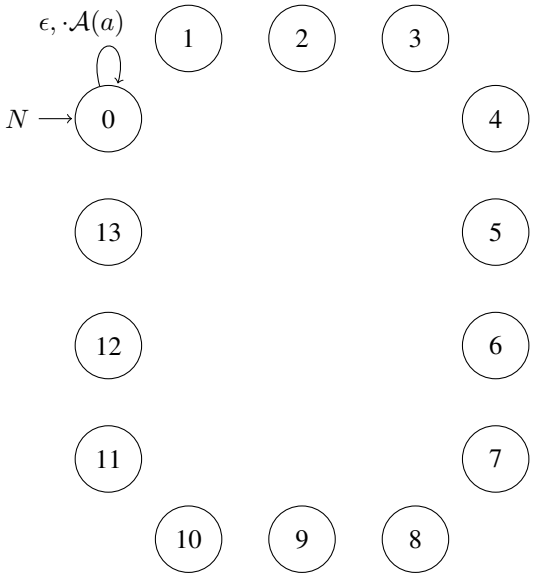


Figure 4. Adaptive automaton N representing notes from the annotated Gregorian staff. Adaptive function $\mathcal{A}$ is presented in Algorithm 1.

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.

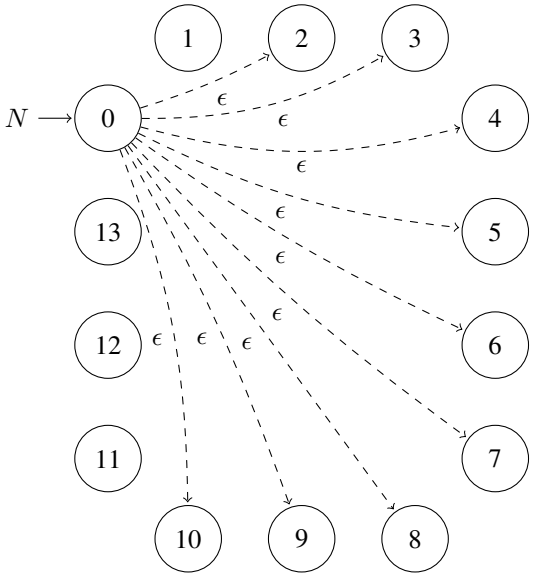


Figure 5. New configuration of the adaptive automaton N (Figure 4) after the execution of $\mathcal{A}(a)$, with $a = 2$, representing the authentic protus mode. Transitions have omitted the associated adaptive function ψ_{rand} due to space constraints.

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

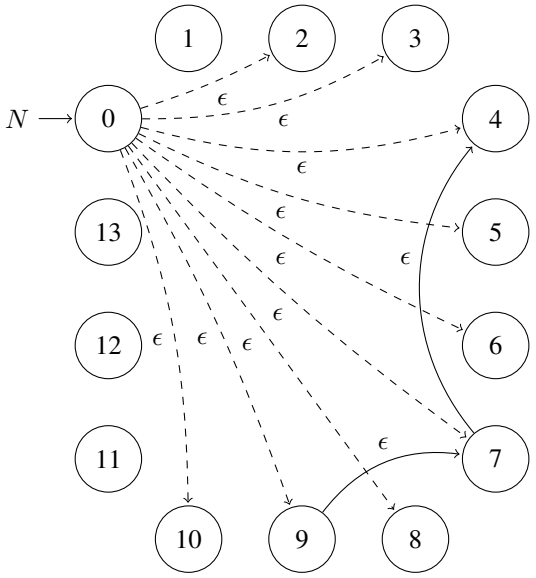


Figure 6. A possible new configuration after the execution of the adaptive function *climacus*(9) within an authentic protus mode (Figure 5).

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:

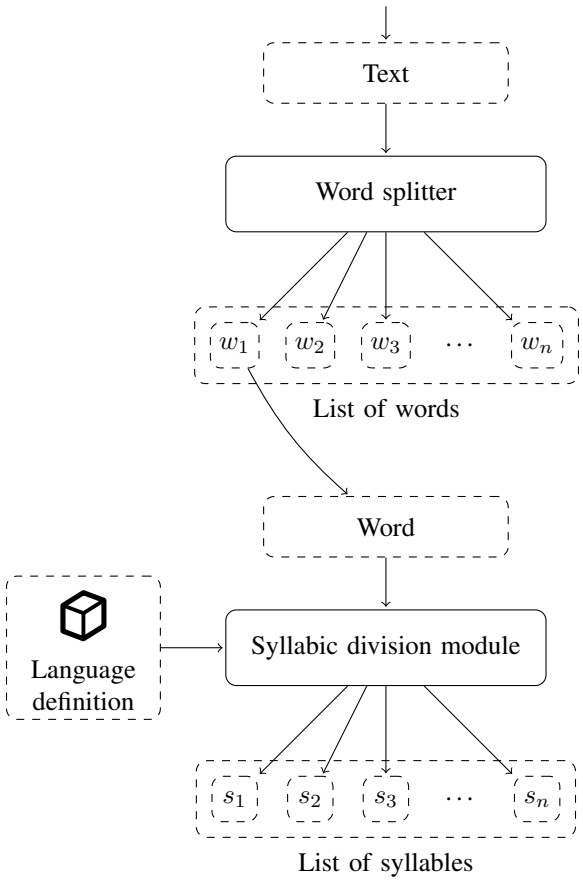


Figure 7. Text manipulation process overview. Note that the module requires a proper language definition containing the syllabic division rules in order to properly process each word.

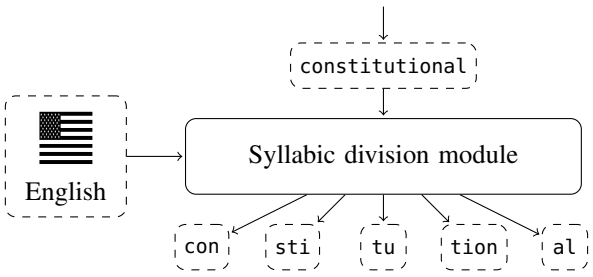


Figure 8. Syllabic division module.

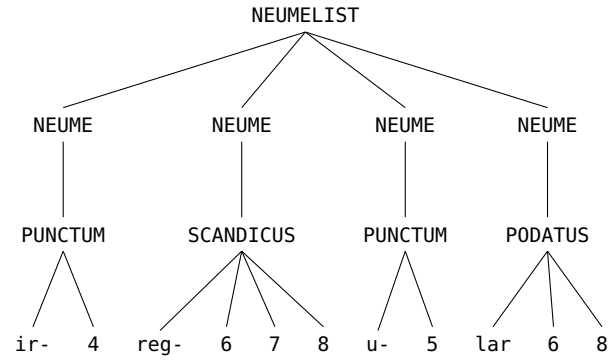


Figure 9. Example of a tree structure containing four neumes and the corresponding syllables for the English word `irregular`. Values are represented in the leaves.

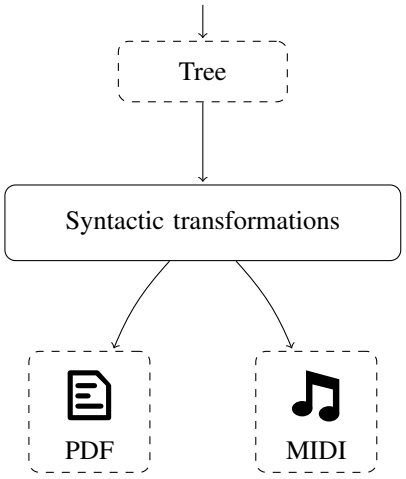


Figure 10. Outline of syntactic transformations applied to the tree, resulting in both PDF and MIDI output formats.

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

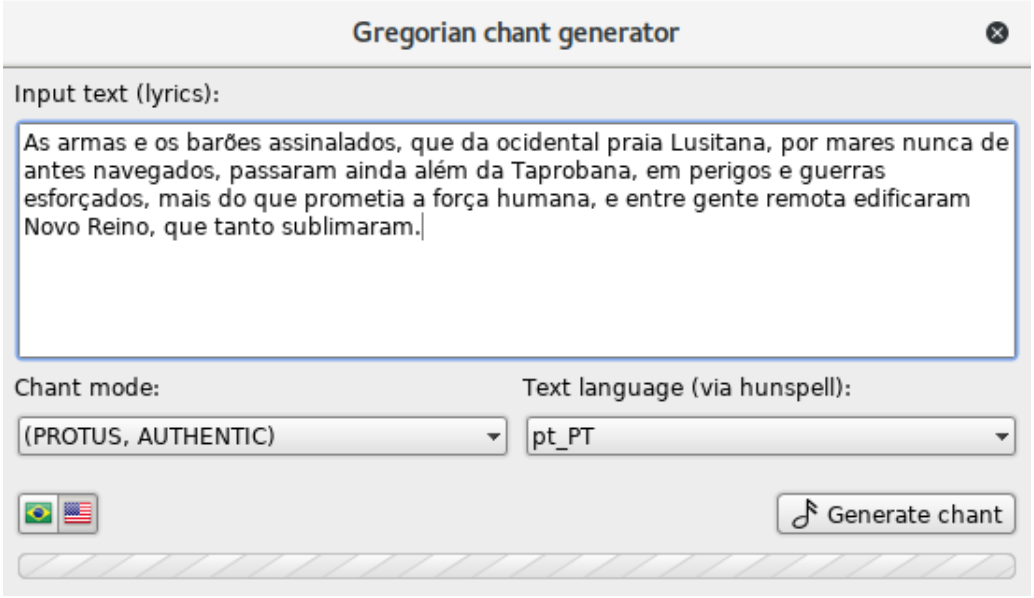


Figure 12. Gregorian chant melody generator interface. It is a simple interface in which we simply type the input text and select both mode and text language.

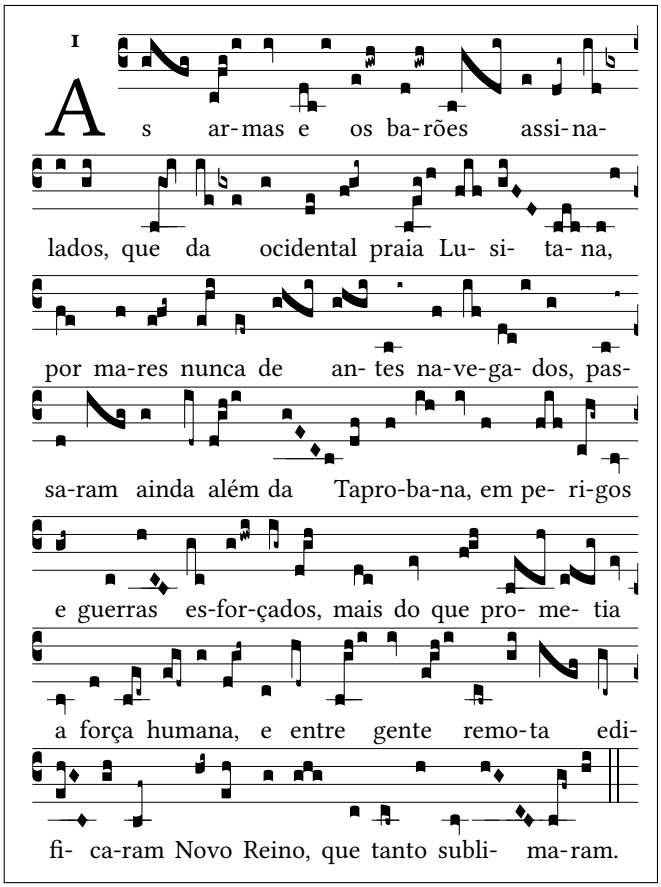


Figure 13. Resulting chant melody score from Canto I (Figure 11).

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.

ADAPBEEN - Adaptible Automata Framework

Reinaldo Felipe Soares Araujo; João Augusto Felberg Jacobsen; Willians Magalhães Primo;
Reginaldo Inojosa da Silva Filho.

***Resumo*— In this paper we present the ADAPBEEN, a Framework that implements the algebraic model of adaptive automata of first and second order. Our framework allows us to build applications that use adaptive technology, that is, programs that change their behavior depending on the received inputs. To this end, ADAPBEEN was built in a modular way. For better performance, it was developed in C ++ and in a way to allow the code to be portable.**

I. INTRODUÇÃO

Atualmente cada vez mais vêm tem sido utilizados dispositivos auto-modificantes em diferentes áreas tais como: Linguagens Naturais [1], Linguagens de Programação [2], Robótica [3], etc. E apesar da teoria adaptativa estar madura, geralmente as implementações estão restritas a projetos específicos [4], [5]. Algumas bibliotecas, ferramentas e frameworks tem sido construídos na tentativa de resolver esse problema, tais como: AdapTools[6], AdapLib[5], e a biblioteca para autômatos adaptativos[7].

Diante dessa demanda por dispositivos auto-modificantes, e por conta das questões relacionadas ao estudo da complexidade nos autômatos adaptativos, foi proposta uma nova formulação algébrica para o autômatos adaptativos. Esta formulação levou à descrição dos autômatos de primeira e segunda ordem aplicada à inferência indutiva de linguagens formais [8], porém a mesma ainda não havia sido implementada e experimentada. Este artigo apresenta o framework ADAPBEEN, que consiste na implementação, em código, dessa formulação algébrica.

O ADAPBEEN é um Framework de codigo aberto portátil, desenvolvido em C++ e que pode ser utilizado no Windows, Linux ou Mac. Ele utiliza os conceitos de camada para a estruturação de sua arquitetura, tornando-o mais didático, modularizado e fiel à teoria. O nosso objetivo em desenvolvê-lo consiste em prover uma biblioteca para o programador que pretenda utilizar a tecnologia adaptativa na construção de soluções computacionais.

O trabalho esta organizado como se segue: A seção seguinte apresenta uma fundamentação teórica sobre dispositivos adaptativos para melhor compreensão da implementação realizada. A seção III refere-se aos detalhes estruturais do ADAPBEEN com uma descrição de cada classe implementada e suas funcionalidades. E as seções IV e V além de uma sucinta explicação sobre os autômatos de primeira e segunda ordem as seções contem exemplos.

*Agradeço ao Grupo PET Fronteira e a Fábrica de Software pelo apoio.

II. FUNDAMENTAÇÃO TEÓRICA

Os autômatos adaptativos possuem a capacidade de modificar a própria estrutura em tempo de execução, com isso, estados e transições podem ser acrescentados ou removidos. Essas alterações estruturais dependem da entrada, do estado atual do autômato e das transições associadas ao mesmo que, caso forem adaptativas, podem alterar a topologia do autômato. Em função desse comportamento, o poder computacional o autômato adaptativo é equivalente ao de uma Máquina de Turing [9], capaz de reconhecer, inclusive, linguagens livres de contexto. Embora o autômato adaptativo seja dotado de grande poder computacional, o mesmo lida com o mesmo problema da Máquina de Turing em relação a incomputabilidade [10].

Durante os últimos anos, a tecnologia adaptativa tem se desenvolvido em uma série de aplicações. Os autômatos adaptativos, uma das facetas da tecnologia adaptativa, consiste em uma solução computacional para problemas complexos. Nosso objetivo neste trabalho é disponibilizar um framework, o ADAPBEEN, que sirva de base para os desenvolvedores criarem suas aplicações adaptativas.

Novas formulações para o modelo são continuamente apresentadas [10], [8]. O modelo escolhido para implementar o ADAPBEEN foi a formulação algébrica para o Autômato Finito Adaptativo de Segunda Ordem, que tem uma forte conexão com o aprendizado indutivo no limite. O formalismo dos autômatos finitos de segunda ordem é desenvolvido sobre o modelo dos autômatos finitos de primeira ordem, uma extensão natural dos autômatos adaptativos clássicos[8]. Essa nova formulação procura resolver os problemas de ambiguidades e imprecisões das formulações anteriores, além de aumentar a capacidade de expressão do modelo original. Neste contexto surgiram então as transformações adaptativas ocorridas em um autômato, sendo que elas foram conceituadas como ações básicas de inserção e remoção de transições no dispositivo subjacente. Utilizando-se esses operadores, foi definida uma variante da formulação dos autômatos adaptativos (modelo original), chamada autômatos adaptativos de primeira ordem. A escolha do nome "Autômatos Adaptativos de Primeira Ordem"foi proposital pelo autor, para levar a dedução de uma próxima ordem que tem como dispositivo subjacente ou de ordem anterior, ou seja, um autômato adaptativo de segunda ordem que insere e remove transições adaptativas de primeira ordem.

III. ADAPBEEN FRAMEWORK

Como mencionado inicialmente, o ADAPBEEN segue uma hierarquia de camadas que pode ser observada no dia-

grama da Figura 4. A primeira camada é formada pela classe *automaton* e a segunda camada, formada pela classes *DFA* e *NFA*, são responsáveis pelos autômatos finitos determinísticos e não-determinísticos que são capazes de reconhecer linguagens irregulares. A terceira camada é formada pela classe *AAFO* que é capaz de gerar autômatos adaptativos de primeira ordem, capazes de reconhecer linguagens livres de contexto. A última camada é formada pela classe *AASO*, capaz de gerar autômatos adaptativos de segunda ordem, que podem aprender famílias de linguagem.

É importante destacar a sequência com que cada camada é executada, as palavras reservadas para lidar com a adaptatividade e o fato de que as mudanças ocorrem em tempo de execução. A sequência de execução é da primeira camada até a última, ou seja, primeiro serão executadas as transições pertencentes ao automato finito não-determinístico, em seguida as sequências positivas e negativas de primeira ordem associadas a essa transição e por último as sequencias positivas e negativas de segunda ordem que também são associadas a transição. As palavras reservadas podem ser mudadas nas definições e são elas "qs"e -1"o primeiro significa que é um estado dinâmico e a segunda representa o símbolo vazio.

Segue abaixo uma descrição resumida de cada classe e suas funcionalidades:

A. Classe state

Esta classe é um dos elementos básicos de um autômato juntamente com as transições. Os estados são muito importantes em um autômato para se determinar um estágio e a determinada ação a entrada recebida. A classe *state* estrutura os estados de um autômato, os quais podem ser estados estáticos ou dinâmicos.
A classe *state* possui os atributos *_name*, *symbol*, *activated*, *stateOrigin*. Estes atributos são utilizados nos estados dinâmicos quando o autômato é de primeira ou segunda ordem. Os estados estáticos são os estados tradicionais/estáticos usados nos autômatos finitos determinísticos e não determinísticos, sendo necessário para o funcionamento do mesmo somente um nome. Exemplo:

state q1("q1);

Os estados dinâmicos são úteis quando se é necessário referenciar um estado pela função que ele exerce e que pode sequer existir. Eles são constituídos de três parâmetros, que são usados no método de busca *Sch* que está implementado na classe *automaton*.
Exemplo: Supondo que a cadeia de entrada foi “aabb” e que o autômato adaptativo criou um estado chamado *s4* e uma nova transição que liga o estado *s4* com o estado *q3* (o qual é estático) com o caractere ’a’. Ao declarar um estado dinâmico com essa forma: *state* desconhecido("q3","a",0), é feita uma solicitação para que procure um estado que tenha uma transição com *q3* com o símbolo ’a’ e que a transição esteja partindo dele (se o terceiro parâmetro fosse um então estaria chegando nele). Cabe salientar que caso a cadeia de entrada fosse diferente, por exemplo uma cadeia maior, então o estado *s4* continuaria existindo, mas poderia não ser o estado que conecta o estado *q3* ao autômato.

B. Classe transition

A classe *transition* possui os atributos *origin*, *destiny*, *symbol* e *shape* que são os componentes básicos de uma transição. Um autômato, geralmente, é composto em sua estrutura por estados e transições, sendo essa classe responsável pelas transições do autômato.

As transições são elementos básicos de um autômato, que informa para qual estado o autômato deve ir caso esteja em um determinado estado e receba um determinado símbolo como entrada. Nesse modelo as transições podem ser criadas através da classe *transition* que possui os atributos *origin*, *destiny*, *symbol* e *shape*. Além dos *getters* se *setters* usuais do paradigma orientado a objeto, a classe traz também os métodos *Start* e *Fin* que são utilizados no modelo algébrico.

C. Classe automaton

Cada autômato implementado no ADAPBEEN tem em sua base a classe *automaton*, que possui os elementos básicos dos autômatos, sendo eles o alfabeto, estados finais, estados iniciais, transições e o nome do autômato. Na classe *automaton* estão implementados também os métodos padrões da programação objeto como *getters* e *setters*, métodos do modelo teórico, e métodos para visualização. Os métodos do modelo teórico são os *Sch*, *rem* e *ins* que possuem a função de busca, remoção e inserção extremamente úteis para a adaptatividade. E para a visualização os métodos *printStates*, *printInitialStates*, *printFinalStates*, *printTransitions*, *printAlphabet* que possuem a função de imprimir todos os estados do autômato, imprimir os estados definidos como iniciais, imprimir os estados definidos como finais, imprimir as transições pertencentes ao autômato e imprimir os símbolos que o autômato reconhece respectivamente.

D. Classe DFA

A classe *DFA* abreviação de "Deterministic Finit Automata"(Autômato Finito Determinístico), possui o método *output* implementado, que permite saber se uma palavra pertence ou não ao autômato. Se o retorno do método *output* for verdadeiro então significa que a palavra parametrizada pertence ao autômato, se for falsa significa que não pertence ao autômato.

E. Classe NFA

A classe *NFA* abreviação para "Nondeterministic Finite Automaton"(Autômatos Finitos Não-determinísticos) é usada como base para os autômatos adaptativos de primeira e segunda ordem. Esta classe contem os métodos *output*, *toAFNDG* e *toDFA*. O método *output* é utilizado para saber se a palavra parametrizada pertence ou não ao autômato, ou seja, se o retorno for verdadeiro a palavra então pertence ao autômato e se for falso então a palavra não pertence ao autômato. O método *toAFNDG* transforma o autômato em um autômato generalizado que é um autômato que possui apenas um estado final. O método *toDFA* retorna um autômato finito determinístico equivalente.

F. Classe AAFO

Um automato adaptativo de primeira ordem tem como seu dispositivo subjacente um autômato finito não-determinístico nesse modelo. A classe *AAFO* abreviação para "Adaptatible Automata First Order"(Automato Adaptativo de Primeira Ordem) é capaz de lidar com autômatos adaptativos que podem ser reconhecedores de linguagens livres de contexto. A classe tem um novo atributo chamado *setoftests* que é o conjunto de comportamentos do automato de primeira ordem. Os elementos do conjunto de comportamentos é um par que possui um ID e um *pairFO* (pair First Order) que é um par com uma sequência positiva e negativa do que deve ser acrescentado ou removido do dispositivo subjacente. As novos métodos implementados na classe são *insFO*, *remFO* e os *getters* e *setters* relativos ao *setoftests*. Os métodos *insFO* e *remFO* são para remover pares adaptativos de primeira ordem do conjunto de comportamento (*setoftests*). Uma importante observação é que o ID do par adaptativo faz referência a qual transição do dispositivo subjacente ele está relacionado, ou seja, qual será a transição que o ativará. Logo o ID do par adaptativo e da transição necessitam ser correspondentes.

G. Classe pairFO

Um autômato adaptativo de primeira ordem tem como dispositivo subjacente um automato finito não-determinístico nesse Framework, sendo assim sequência positivas e negativas vão adicionar e remover transições e estados do *afnd* subjacente. A classe *pairFO* implementa o par ordenado adaptativo de transições próprias e estrangeiras que devem ser removidas e acrescentadas do automato. A definição de transições próprias e estrangeiras está disponível no modelo teórico na qual foi baseado o Framework.

H. Classe negativeSequence

Sequências negativas são transições estrangeiras que devem ser acrescentadas ao automato. Aqui representada pela classe *negativeSequence*.

I. Classe positiveSequence

Sequências positivas são transições próprias que devem ser removidas do autômato aqui representada pela classe *positiveSequence*.

J. Classe AASO

Um autômato adaptativo de segunda ordem tem como dispositivo subjacente um autômato adaptativo de primeira ordem, ou seja, um autômato adaptativo de segunda ordem adiciona e remove pares adaptativos de primeira ordem. Uma vez que um autômato adaptativo de primeira ordem pode ser um reconhecedor para uma linguagem livre de contexto ao modificar o automato de primeira ordem, a linguagem do reconhecedor é alterado podendo reconhecer uma nova linguagem. A classe *AASO* é a camada responsável pela adaptatividade de segunda ordem com capacidade de inferir um autômato de primeira ordem aprendendo assim uma linguagem. *AASO* tem três novos atributos chamados de *setoftestsSO*, *inferedAutomata* e *flag*. O atributo *setoftestsSO*, semelhante ao *setoftests* da classe *AAFO*, é o conjunto de comportamentos do *AASO* sendo seus elementos pares de inteiros e pares adaptativos de segunda ordem, isto é, o inteiro é o ID dando referência a transição que ativa as adaptações que devem ser feitas e os pares adaptativos de segunda ordem são os *pairSO* que são sequencias positivas e negativas de pares adaptativos de primeira ordem que devem ser incluídos e removidos do *AASO*. Além disso o atributo *inferedAutomata* é atualizado após um automato ser inferido pelo método *inferAutomata*, sendo assim capaz de agir como um reconhecedor para a linguagem inferida. Os novos métodos implementados na classe são o *inferAutomata*, *insSO* e *remSO*. O método *inferAutomata* cada vez que é utilizado transformações são feitas no dispositivo subjacente (*inferedAutomata*) que após se estabilizar atualiza o atributo *flag* para o valor *true* significando que ouve uma convergência e foi aprendida uma linguagem.

K. Classe pairSO

Um par adaptativo de segunda ordem é um par que possui uma sequência positiva e uma sequência negativa que são os pares adaptativos de primeira ordem, que devem ser removidos e acrescentados ao autômato de primeira ordem. A classe *pairSO* implementa o par ordenado de segunda ordem que possuem pares ordenados de primeira ordem, que devem ser removidos ou acrescentados ao autômato modificando assim o mesmo.

L. Classe negativeSequenceSO

Sequência negativa de segunda ordem são os pares adaptativos de primeira ordem que devem ser acrescentados ao autômato. Neste Framework a sequência negativa de segunda ordem é implementada como classe *negativeSequenceSO*.

M. Classe positiveSequenceSO

Sequência positiva de segunda ordem são os pares adaptativos de primeira ordem que devem ser removidos do autômato. Aqui representada pela classe *positiveSequenceSO*.

IV. O Autômato Adaptativo de Primeira Ordem

Um autômato adaptativo de primeira ordem que é capaz de reconhecer linguagens livres de contexto, e como já fora dito na seção III, a terceira camada representada pela classe AAFO (seção III-F) é capaz de criar autômatos adaptativos de primeira ordem. Segue abaixo (Figura VI) um exemplo de implementação de um autômato de primeira ordem capaz de reconhece a linguagem $L1 = a^n b^n \mid n \geq 1$ ilustrado na Figura V.

V. O Autômato Adaptativo de Segunda Ordem

A quarta camada do ADAPBEEN é capaz de inferir autômatos adaptativos de primeira ordem e está implementada na classe AASO descrita na seção III-J. Que foi proposto no modelo teórico[8] ele deve ser capaz de aprender uma linguagem, uma discussão e experimento para um próximo trabalho. Segue abaixo na Figura V um exemplo de um Autômato Finito Adaptativo de Segunda Ordem que pode

inferir um reconhecedor para as linguagens $L1 = a^n b^n c$ e $L2 = ab^n c^n$ e consequentemente aprender um linguagem. A Figura V exemplifica um automato de primeira ordem inferido pelo AASO(Figura V) e capaz de reconhecer a linguagem $L1 = a^n b^n c$.

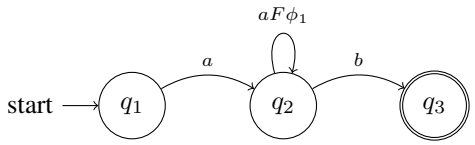


Fig. 1. Automato que reconhece a $L1 = a^n b^n \mid n \geq 1$

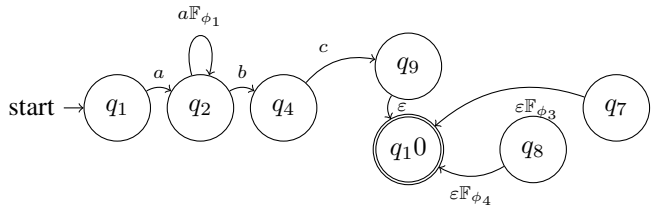


Fig. 2. Exemplo de autômato inferido pelo autômato da Figura V que reconhece a linguagem $L2 = ab^n c^n \mid n \geq 1$

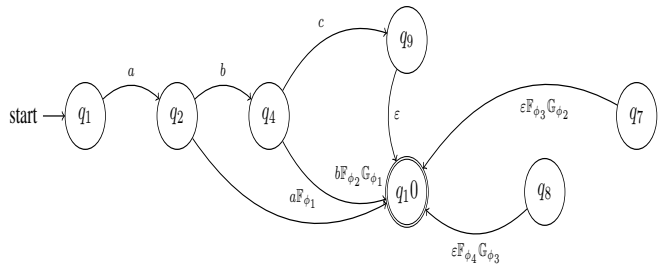


Fig. 3. Exemplo de Autômato Finito de Segunda Ordem capaz de aprender as linguagens $L1 = a^n b^n c$ e $L2 = ab^n c^n \mid n \geq 1$

VI. Conclusões

A implementação do ADAPBEEN é uma iniciativa que visa divulgação e promoção da tecnologia adaptativa. A partir deste trabalho, será possível aos programadores o desenvolvimento de soluções adaptativas de maneira rápida e robusta. O framework apresentado aqui é acessivel a todos que desejam utilizá-lo e estará disponível no BitBucket.

O desenvolvimento desse framework foi um trabalho realizado no âmbito do grupo de pesquisa BioCompTech. O desenvolvimento seguiu naturalmente o modelo algébrico e permite, a quem o utilizar, o acesso à adaptatividade de primeira e segunda ordem.

Os teste realizados com o framework mostraram que o ADAPBEEN encontra-se completamente operacional. Como atividades futuras, planeja-se a criação de aplicativos adaptativos nas áreas de processamento de linguagens e jogos.

Referências

[1] D. Padovani and J. J. Neto, "Adaptive automata applied to natural language processing," *Procedia Computer Science*, vol. 109, pp. 1152–1157, 2017.

[2] P. Cereda and J. J. Neto, "Um arcabouço para extensibilidade em linguagens de programação," in *Memórias do WTA 2015 IX Workshop de Tecnologia Adaptativa*, p. 18.

[3] L. C. Barros and A. R. Hirakawa, "An approach by straight line segment adaptive techniques in robot navigation," *IEEE Latin America Transactions*, vol. 12, no. 7, pp. 1292–1297, 2014.

[4] H. Pistori, J. J. Neto, and E. Costa, "A free software for the development of adaptive automata," in *Proceedings of the IV Workshop on Free Software-WSL (IV International Forum on Free Software)*, 2003.

[5] F. L. Siqueira, "The adaplib library for execution of adaptive devices: Architecture and use," *IEEE Latin America Transactions*, vol. 9, no. 2, pp. 213–218, 2011.

[6] L. Jesus, D. Santos, A. Castro Jr, and H. Pistori, "Adaptools 2.0: Aspectos de implementação e utilização," *Revista IEEE América Latina. Vol*, vol. 5, pp. 1548–0992, 2007.

[7] N. L. Werneck, "Biblioteca para autômatos adaptativos com regras de transição armazenadas em objetos feita em c+," in *Segundo Workshop de Tecnologia Adaptativa*, 2008, pp. 22–26.

[8] R. I. d. Silva Filho, "Uma nova formulação algébrica para o autômato finito adaptativo de segunda ordem aplicada a um modelo de inferência indutiva." Ph.D. dissertation, Universidade de São Paulo, 2011.

[9] R. L. A. Rocha and J. J. Neto, "Autômato adaptativo, limites e complexidade em comparação com máquina de turing," in *Proceedings of the second Congress of Logic Applied to Technology–LAPTEC*, 2000, pp. 33–48.

[10] D. Queiroz, "Uma definição simplificada para o estudo das propriedades dos autômatos finitos adaptativos," in *Quarto Workshop de Tecnologia Adaptativa*, 210.

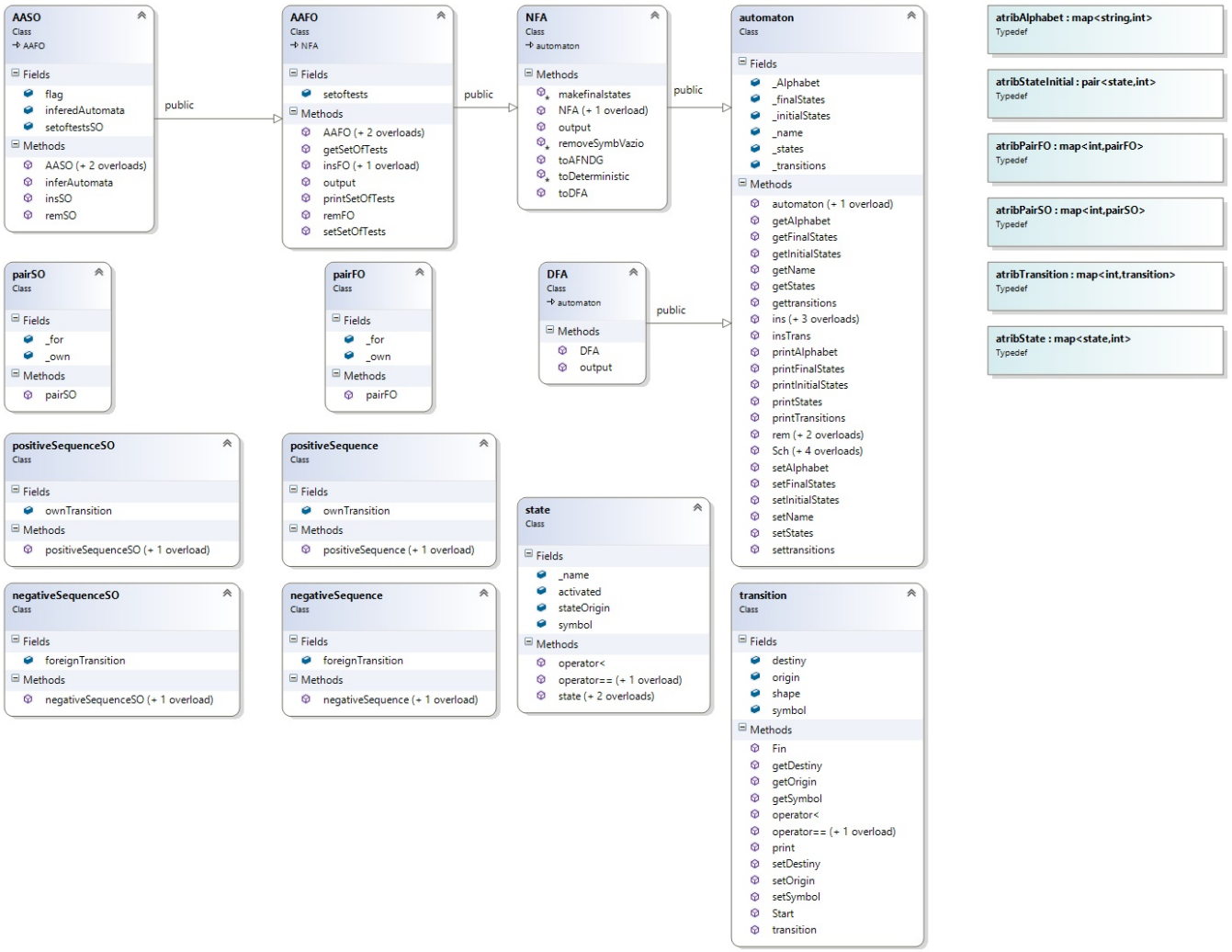


Fig. 4. Estrutura do framework.

```
#include <string>
#include <iostream>
#include <map>
#include "AASO.h"

using namespace std;

int main() {
        atribAlphabet Alfabeto;
        atribState Estados;
        atribTransition mapTransicoes;
        atribState estadosIniciais;
        atribState estadoFinais;
        Alfabeto.insert(make_pair("a", 0));
        Alfabeto.insert(make_pair("b", 1));
        Estados.insert(make_pair(state("q1"), 0));
        Estados.insert(make_pair(state("q2"), 1));
        Estados.insert(make_pair(state("q3"), 2));
        mapTransicoes.insert(make_pair(0, transition(state("q1"), "a", state("q2"))));
        mapTransicoes.insert(make_pair(1, transition(state("q2"), "a", state("q2"))));
        mapTransicoes.insert(make_pair(2, transition(state("q2"), "b", state("q3"))));
        estadosIniciais.insert(make_pair(state("q1"), 0));
        estadoFinais.insert(make_pair(state("q3"), 2));

        //map <int, transition>
        atribTransition _pro;
        atribTransition _for;

        positiveSequence transPro2;
        negativeSequence transFor2;

        atribPairFO conjuntodeComportamentosPO;
        atribPairFO auxPO;

        _pro.insert(make_pair(_pro.size(), transition(state("q3", "b", 0), "b", state("q3"))));
        _for.insert(make_pair(_for.size(), transition(state("q3", "b", 0), "b", state("qs"))));
        _for.insert(make_pair(_for.size(), transition(state("qs"), "b", state("q3"))));

        transPro2 = positiveSequence(_pro);
        transFor2 = negativeSequence(_for);

        pairFO novoPO(transPro2, transFor2);
        //pairFO novoPO;
        pair < int, pairFO > F1(1, novoPO);

        novoPO = pairFO(transPro2, transFor2);
        conjuntodeComportamentosPO.insert(F1);
        AAFO teste("AAFO", Alfabeto, Estados, estadosIniciais, estadoFinais, mapTransicoes, conjuntodeComportamentosPO);

        string entrada;
        while (cin >> entrada) {
                if (teste.output(entrada)) {
                        cout << "Aceita" << endl;
                }
                else {
                        cout << "Nao aceita" << endl;
                }
        }
}
```

Fig. 5. Exemplo de implementação de um Autômato Finito Adaptativo de Primeira Ordem que reconhece a $L1 = a^n b^n \mid n \geq 1$

Etiquetamento Morfológico Usando Tecnologia Adaptativa

D. Padovani, A. T. Contier e J. José Neto

***Resumo*— Este trabalho apresenta uma revisão dos conceitos de Tecnologia Adaptativa e de Processamento da Linguagem Natural, e descreve a arquitetura do reconhecedor gramatical Linguístico, com ênfase no módulo Identificador Morfológico. Em seguida, são apresentados experimentos nos quais as submáquinas adaptativas de etiquetamento morfológico foram avaliadas, comparando seu desempenho ao do etiquetamento realizado sem o uso de tecnologia adaptativa. Por fim, são apresentadas as conclusões dos experimentos e as linhas de continuidade da pesquisa.**

***Palavras Chave* — Autômatos Adaptativos, Processamento de Linguagem Natural, Reconhecedores Gramaticais, Gramáticas Livres de Contexto, Gramáticas Adaptativas.**

I. Autômatos Adaptativos

O AUTÔMATO adaptativo é uma máquina de estados à qual são impostas sucessivas alterações resultantes da aplicação de ações adaptativas associadas às regras de transições executadas pelo autômato [1]. Dessa maneira, estados e transições podem ser eliminados ou incorporados ao autômato em decorrência de cada um dos passos executados durante a análise da entrada. De maneira geral, pode-se dizer que o autômato adaptativo é formado por um dispositivo convencional, não adaptativo, e um conjunto de mecanismos adaptativos responsáveis pela auto modificação do sistema.

O dispositivo convencional pode ser uma gramática, um autômato, ou qualquer outro dispositivo que respeite um conjunto finito de regras estáticas. Este dispositivo possui uma coleção de regras, usualmente na forma de cláusulas if-then, que testam a situação corrente em relação a uma configuração específica e levam o dispositivo à sua próxima situação. Se nenhuma regra é aplicável, uma condição de erro é reportada e a operação do dispositivo, descontinuada. Se houver uma única regra aplicável à situação corrente, a próxima situação do dispositivo é determinada pela regra em questão. Se houver mais de uma regra aderente à situação corrente do dispositivo, as diversas possíveis situações seguintes são tratadas em paralelo e o dispositivo exibirá uma operação não determinística. Os mecanismos adaptativos são formados por três tipos de ações adaptativas elementares: consulta (inspeção do conjunto de regras que define o dispositivo), exclusão (remoção de alguma regra) e inclusão (adição de uma nova regra).

Autômatos adaptativos apresentam forte potencial de aplicação ao processamento de linguagens naturais, devido à facilidade com que permitem representar fenômenos linguísticos complexos tais como dependências de contexto. Adicionalmente, podem ser implementados como um formalismo de reconhecimento, o que permite seu uso no pré-processamento de textos para diversos usos, tais como: análise sintática, verificação de sintaxe, processamento para traduções automáticas, interpretação de texto, corretores gramaticais e base para construção de sistemas de busca semântica e de aprendizado de línguas auxiliados por computador.

Diversos trabalhos confirmam a viabilidade prática da utilização de autômatos adaptativos para processamento da linguagem natural. É o caso, por exemplo, de [2], que mostra a utilização de autômatos adaptativos na fase de análise sintática; [3] que apresenta um método de construção de um analisador morfológico e [4], que apresenta uma proposta de autômato adaptativo para reconhecimento de anáforas pronominais segundo algoritmo de Mitkov.

II. Processamento da Linguagem Natural: Revisão da Literatura

O processamento da linguagem natural requer o desenvolvimento de programas que sejam capazes de determinar e interpretar a estrutura das sentenças em muitos níveis de detalhe. As linguagens naturais exibem um intricado comportamento estrutural visto que são profusos os casos particulares a serem considerados. Uma vez que as linguagens naturais nunca são formalmente projetadas, suas regras sintáticas não são simples nem tampouco óbvias e tornam, portanto, complexo o seu processamento computacional. Muitos métodos são empregados em sistemas de processamento de linguagem natural, adotando diferentes paradigmas, tais como métodos exatos, aproximados, pré-definidos ou interativos, inteligentes ou algorítmicos [5]. Independentemente do método utilizado, o processamento da linguagem natural envolve as operações de análise léxico-morfológica, análise sintática, análise semântica e análise pragmática [6]. A análise léxico-morfológica procura atribuir uma classificação morfológica a cada palavra da sentença, a partir das informações armazenadas no léxico [7]. O léxico ou dicionário é a estrutura de dados contendo os itens lexicais e as informações correspondentes a estes itens. Entre as informações associadas aos itens lexicais, encontram-se a categoria gramatical do item, tais como substantivo, verbo e adjetivo, e os valores morfossintático-semânticos, tais como gênero, número, grau, pessoa, tempo, modo, regência verbal ou nominal. Na etapa de análise sintática, o analisador verifica se uma sequência de palavras constitui uma frase válida da língua, reconhecendo-a ou não. O analisador sintático faz uso de um léxico e de uma gramática, que define as regras de combinação dos itens na formação das frases. Nos casos nos quais há a necessidade de interpretar o significado de um

texto, a análise léxico-morfológica e a análise sintática não são suficientes, sendo necessário realizar um novo tipo de operação, denominada análise semântica [7]. Na análise semântica procura-se mapear a estrutura sintática para o domínio da aplicação, fazendo com que a estrutura ganhe um significado. O mapeamento é feito identificando as propriedades semânticas do léxico e o relacionamento semântico entre os itens que o compõe [8]. Já a análise pragmática procura reinterpretar a estrutura que representa o que foi dito para determinar o que realmente se quis dizer. Inserem-se nessa categoria as relações anafóricas, correferências, determinações, focos ou temas, dêiticos e elipses [9]. Em [10] são apresentados trabalhos de pesquisas em processamento de linguagem natural para a Língua Portuguesa, tais como o desenvolvido pelo Núcleo Interinstitucional de Linguística Aplicada (NILC) no desenvolvimento de ferramentas para processamento de linguagem natural; o projeto VISL – Visual Interactive Syntax Learning, sediado na Universidade do Sul da Dinamarca, que engloba o desenvolvimento de analisadores morfossintáticos para diversas línguas, entre as quais o português; e o trabalho de resolução de anáforas desenvolvido pela Universidade de Santa Catarina. A tecnologia adaptativa também tem contribuído com trabalhos em processamento da linguagem natural. Em [11], são apresentadas algumas das pesquisas desenvolvidas pelo Laboratório de Linguagens e Tecnologia Adaptativa da Escola Politécnica da Universidade de São Paulo: um etiquetador morfológico, um estudo sobre processos de análise sintática, modelos para tratamento de não determinismos e ambiguidades, e um tradutor texto voz baseado em autômatos adaptativos.

I. O Reconhecedor Gramatical Linguístico

O Linguístico é uma proposta de reconhecedor gramatical composto de cinco módulos sequenciais que realizam cada qual um processamento especializado, enviando o resultado obtido para o módulo seguinte, tal como ocorre em uma linha de produção, até que o texto esteja completamente analisado [12]. A Fig. 1 apresenta a estrutura geral do Linguístico.

Este artigo procura focar o terceiro módulo, chamado Identificador Morfológico, detalhando sua arquitetura e apresentando um experimento no qual o módulo é testado com *tokens* da Língua Portuguesa, comparando os resultados com os obtidos sem o uso de tecnologia adaptativa. O Identificador Morfológico está diretamente relacionado a dois outros módulos do Linguístico, o Sentenciador e o Tokenizador. O Sentenciador é o módulo que recebe textos e os dividem em sentenças, usando, para isso, expressões regulares que aplicam regras de pontuação e de desambiguação de palavras abreviadas e de palavras compostas. O Tokenizador é o módulo que recebe as sentenças identificadas pelo Sentenciador e as divide em *tokens*, considerando, neste processo, abreviaturas, valores monetários, horas e minutos, numerais arábicos e romanos, palavras compostas, nomes próprios, caracteres especiais e de pontuação final. Os *tokens* são armazenados em estruturas de dados (*arrays*) e enviados um a um para o Identificador Morfológico, que é o módulo do Linguístico responsável pelo etiquetamento morfológico dos *tokens*, usando para isso, tecnologia adaptativa.

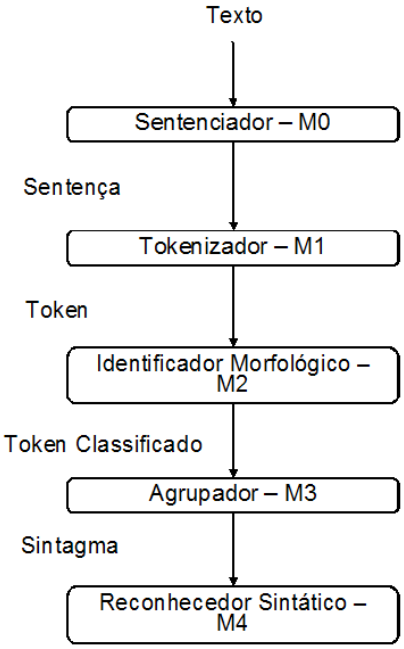


Figura 1. Estrutura do Linguístico.

O Identificador Morfológico é composto por um Autômato Mestre e um conjunto de submáquinas especialistas que acessam bases de dados de classificações morfológicas (Fig.2.1). A prioridade é obter as classificações de corpus já analisados e etiquetados; caso o termo procurado não seja encontrado, o Identificador Morfológico procura por verbos, substantivos e adjetivos, no formato finito e infinito (flexionados e não flexionados), através de uma submáquina de formação e identificação de palavras; por fim, o Identificador Morfológico procura por termos invariáveis, ou seja, termos cuja classificação morfológica é considerada estável pelos linguistas, tais como, conjunções, preposições e pronomes.

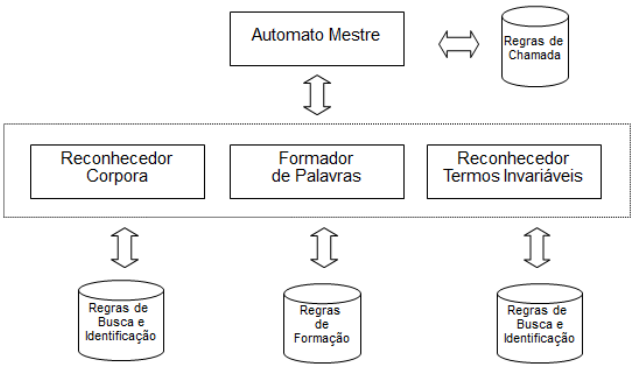


Figura 2.1. Arquitetura do Identificador Morfológico.

O Autômato Mestre (Fig.2.2) é responsável pelo sequenciamento das chamadas às submáquinas, de acordo com um conjunto de regras cadastradas em base de dados. Ele inicia o processamento recebendo o *token* da coleção de *tokens*. Em

seguida, o autômato se modifica através de uma função adaptativa, criando uma submáquina de processamento (M1), uma transição entre o estado 1 e M1, e uma transição que aguarda o estado final da submáquina M1. O token é passado para M1 e armazenado em uma pilha. Quando M1 chega ao estado final, o autômato se modifica novamente, criando uma nova submáquina M2, o estado 2 e as transições correspondentes. Caso o estado final de M1 seja de aceitação, o processo é finalizado no estado 2, caso contrário a máquina M2 é chamada, passando o token armazenado na pilha.

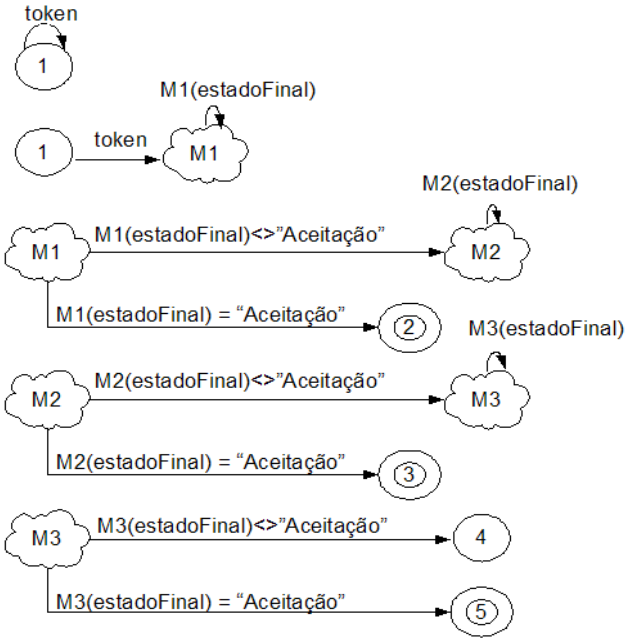


Figura 2.2. Estrutura Adaptativa do Autômato Mestre.

O processo se repete quando M2 chega ao final do processamento, com a criação da submáquina M3, do estado 3 e das transições correspondentes. Um novo ciclo se repete, e se o estado final de M3 é de aceitação, o autômato transiciona para o estado 5, de aceitação, caso contrário, ele vai para o estado 4 de não aceitação. M1, M2 e M3 representam, respectivamente, as submáquinas do Reconhecedor de Corpus, do Formador de Palavras e do Reconhecedor de Termos Invariáveis. Portanto, caso o Autômato Mestre encontre a classificação morfológica ao final de M1, ele não chama M2; caso encontre em M2, não chama M3 e, caso também não encontre em M3, ele informa aos demais módulos do Linguístico que não há classificação morfológica para o termo analisado.

As submáquinas M1, M2 e M3 também foram projetadas de acordo com a tecnologia adaptativa. A submáquina M1 usa um autômato adaptativo que se automodifica de acordo com o tipo de token que está sendo analisado: palavras simples, palavras compostas, números, valores e símbolos. A Fig. 2.3 apresenta a estrutura adaptativa de M1. Inicialmente o autômato é composto por um único estado e por transições para ele mesmo (Fig.2.3, à esquerda). Ao identificar o tipo de token que será analisado (obtido no processo de tokenização), o autômato cria submáquinas e as transições correspondentes.

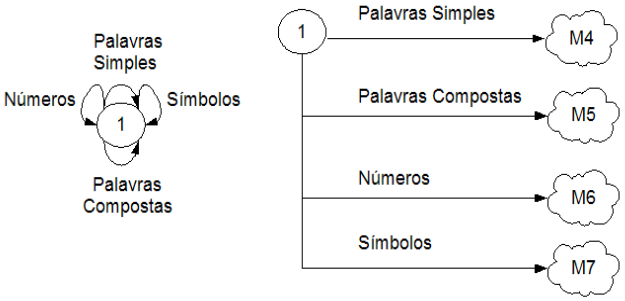


Figura 2.3. Estrutura Adaptativa do Reconhecedor de Corpus M1.

As alternativas de configuração são apresentadas na Fig.2.3, à direita. As submáquinas M4, M5, M6 e M7 reconhecem os tokens através de outro tipo de autômato que os processam sequencialmente de acordo com as letras, números e caracteres especiais que os compõem (Fig. 2.4).

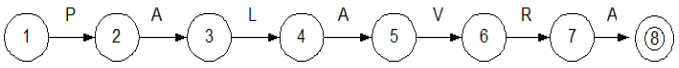


Figura 2.4. Reconhecedor de Palavras Simples.

No exemplo apresentado na Fig.2.4, o token "Palavra" é processado pela máquina M4; se o processamento terminar em um estado de aceitação, o token é reconhecido. As submáquinas M4, M5, M6 e M7 são criadas previamente por um programa que lê Corpus e os convertem em autômatos finitos determinísticos. A estrutura de identificação morfológica é composta por um par [chave, valor], no qual a chave é a classificação morfológica e o valor é o estado de aceitação do elemento lexical. No exemplo apresentado, a chave do item lexical "palavra" seria a classificação morfológica, N - substantivo e o estado "8" faria parte do conjunto de estados de aceitação, indicando que o *token* "palavra" seria sido aceito.

O Formador de Palavras, submáquina M2, também é montado previamente por um programa construtor, levando em consideração as regras de formação de palavras do Português do Brasil, descritas por Margarida Basílio em [13]. A submáquina M2 utiliza o vocabulário do TeP2.0[14], composto por verbos, substantivos e adjetivos, e o conjunto de regras de prefixação, sufixação e regressão verbal descrito pela autora para construir novas formas. No entanto, são necessários alguns cuidados para evitar a criação de estruturas que aceitem palavras inexistentes. A Fig. 2.5 apresenta um exemplo de autômato no qual são aplicados os prefixos "a" e "per", e os sufixos "ecer" e "ar" (derivação parassintética) ao radical "noit" do substantivo noite, que faz parte do TeP2.0.

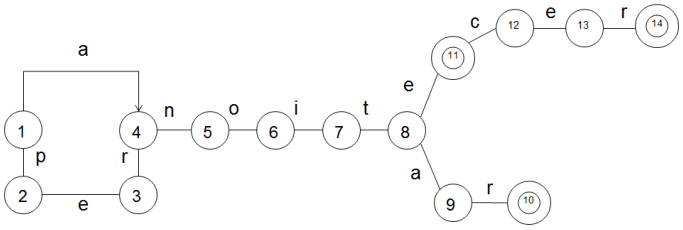


Figura 2.5. Autômato Formador de Palavras.

No exemplo apresentado, as palavras “anoitecer”, “pernoitar” e “anoitar” (sinônimo de “anoitecer”) existem no léxico do português do Brasil. Já pernoitecer é uma combinação que não existe na língua portuguesa. Margarida Basílio diz que algumas combinações não são aceitas simplesmente porque já existem outras construções consagradas pelo uso. Para reduzir o risco de aceitar derivações inexistentes, o processo de construção do autômato restringe as possíveis formações, utilizando apenas as regras que a autora destaca como sendo mais prováveis. É o caso de nominalização de verbos com o uso dos sufixos –ção , –mento e – da. A autora acrescenta que o sufixo –ção é responsável por 60% das formações regulares, enquanto o sufixo –mento é responsável por 20% destas formações. Já o sufixo –da é, via de regra, usado em nominalizações de verbos de movimento, tais como, entrada, saída, partida, vinda, etc.

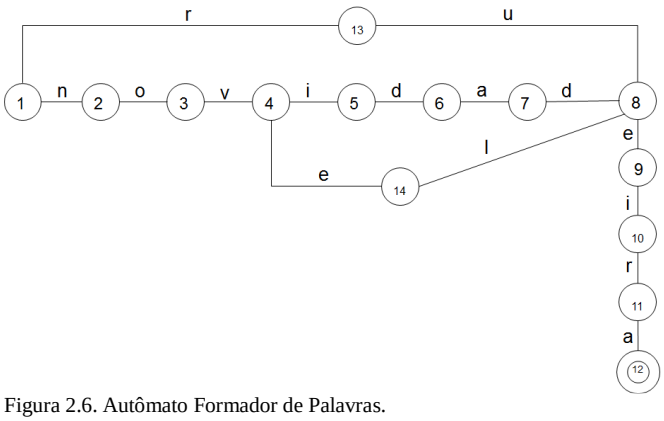


Figura 2.6. Autômato Formador de Palavras.

Outra característica do construtor do autômato é representar os prefixos e sufixos sempre pelo mesmo conjunto de estados e transições, evitando repetições que acarretariam o consumo desnecessário de recursos computacionais. A Fig. 2.6 apresenta exemplo de palavras formadas por reutilização de estados e transições na derivação das palavras rueira, novidadeira e noveleira. Os estados e transições usados para representar o sufixo “eira”, usado para designar são os mesmos nas três derivações.

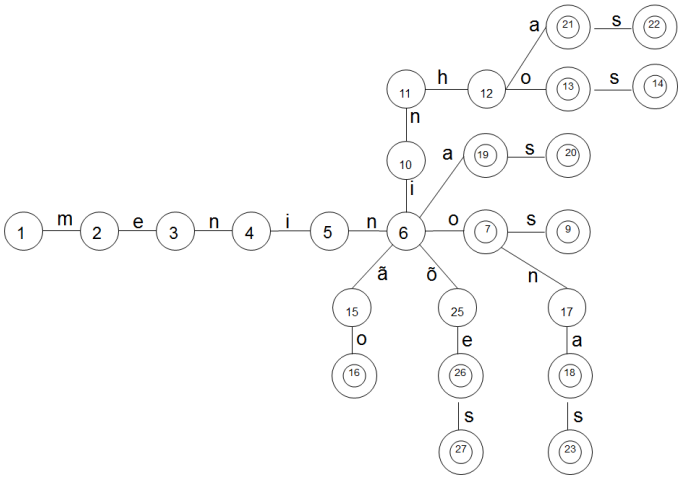


Figura 2.7. Autômato Formador de Flexões Nominais.

A submáquina M2 também reconhece palavras flexionadas, obtidas, no caso de substantivos e adjetivos, através da aplicação de sufixos indicativos de gênero, número e grau aos radicais do vocabulário TeP2.0. A Fig. 2.7 apresenta um exemplo de autômato usado para a formação das formas flexionadas do substantivo menino. Foram adicionados ao radical “menin” as flexões “o(s)”, “a(s)”, “ão”, “ões”, “onona(s)”, “inho(s)” e“inha(s)”.

No caso de verbos, foi criada uma estrutura de estados e transições para representar as flexões de tempo, modo, voz e pessoa, obtendo-se, assim, as respectivas conjugações.

A Fig. 2.8 apresenta um exemplo de autômato usado para a formação das formas flexionadas do presente do indicativo do verbo andar. Foram adicionados ao radical “and” as flexões “o”, “as”, “a”, “andamos”, “ais” e “andam”.

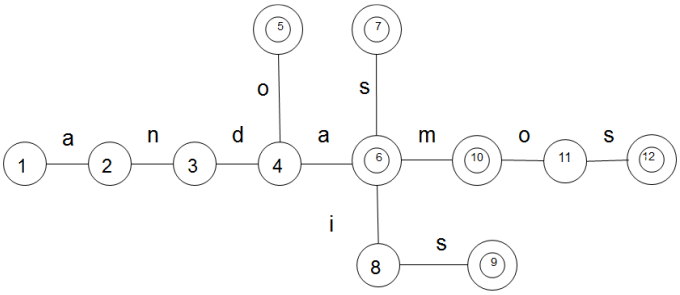


Figura 2.8. Autômato Formador de Flexões Verbais.

A estrutura de armazenamento da submáquina M2 também é composta por um par [chave, valor], no qual a chave é a classificação morfológica e o valor é o estado de aceitação do elemento lexical. Já a submáquina M3 é um autômato que varia em função do tipo de termo (conjunções, preposições e pronomes) e utiliza uma estrutura arbórea similar a M4. A Fig. 2.9 apresenta um exemplo de autômato usado no reconhecimento de termos deste domínio. A estrutura de armazenamento da submáquina M3 é composta por um par [chave, valor], no qual a chave é a classificação morfológica e o valor é o estado de aceitação do elemento lexical.

Figura 2.9. Autômato Reconhecedor de Conjunções, Preposições e Pronomes.

Como as palavras podem ter mais do que uma classificação morfológica, é necessário utilizar uma técnica para selecionar aquela que é mais apropriada para o contexto analisado. Por exemplo, a palavra “casa” pode ser classificada como substantivo comum, feminino, singular ou como verbo flexionado na 3ª pessoa do singular no tempo presente, modo indicativo. No entanto, tendo em vista o contexto em que palavra se encontra, é possível escolher a classificação mais provável. Por exemplo, se a palavra “casa” vier precedida de um artigo definido, é mais provável que ela seja um substantivo; já se a palavra antecessora for um substantivo próprio, é mais provável que “casa” seja um verbo.

O Automato Mestre utiliza o algoritmo Viterbi [19] para fazer as desambiguações. Este algoritmo encontra a sequência mais provável de etiquetas para uma determinada sequência de *tokens*. A Figura 2.10 apresenta um exemplo da aplicação do

algoritmo Viterbi para desambiguação do *token* "casa" na sequência "A casa". O algoritmo recebe como parâmetro de entrada o *token* que está sendo analisado e encontra a etiqueta mais provável, considerando a sequência de etiquetas anteriores e as possíveis etiquetas do *token* analisado.

		A	casa
Etiqueta	<I>	Artigo	?
Opção 1	<I>	Artigo	Substantivo
Opção 1	<I>	Artigo	P(Substantivo|Artigo,<I>)*E(casa|Substantivo)
Opção 2	<I>	Artigo	Verbo
Opção 2	<I>	Artigo	P(Verbo|Artigo,<I>)*E(casa|Verbo)
Resultado	<I>	Artigo	Substantivo - Maior probabilidade

Onde:

<I> = etiqueta que identifica início de frase
P = probabilidade da etiqueta avaliada, considerando as duas últimas
E = probabilidade do *token*, considerando a etiqueta avaliada

Figura 2.10. Exemplo de Aplicação do Algoritmo Viterbi.

II. EXPERIMENTOS E RESULTADOS COMPARATIVOS

O Identificador Morfológico foi desenvolvido em linguagem Python, com o apoio do NLTK [16], uma biblioteca de processamento de linguagem natural. As submáquinas M1, M2 e M3 foram criadas a partir do vocabulário TeP2.0. No caso de verbos, foram implementados estados e transições para representar as flexões de tempo, modo, voz e pessoa, obtendo-se, assim, as respectivas conjugações. No caso dos substantivos e dos adjetivos, foram implementados estados e transições para representar as flexões de número.

O algoritmo Viterbi foi implementado através de um mecanismo de *fall-back*, iniciando a análise com trigramas, passando por bigramas e unigramas, e assumindo a classificação mais comum do corpus, que é substantivo, no caso de não encontrar nenhuma classificação. O mecanismo foi inicialmente treinado com 70% do corpus CINTIL Treebank [17], ficando os 30% restantes para testes. O tamanho do corpus de treino foi, então, progressivamente ampliado, para 75%, 80%, 85% e 90% do corpus CINTIL, deixando sempre o restante para testes. Os resultados foram apurados através do indicador de acurácia (Total de etiquetas corretas/Total de etiquetas *gold standard*) e os resultados são apresentados na Tab. 2.

Corpus	70%	75%	80%	85%	90%
Acurácia	90,69%	90,73%	90,93%	91,02%	91,41%

Tabela 2. Resultados do Etiquetador sem Tecnologia Adaptativa

Em seguida, as submáquinas M1, M2 e M3 foram incorporadas ao mecanismo de *fall-back* de forma a ser chamadas após o processamento dos unigramas e antes do mecanismo de *fall-back* assumir a classificação mais comum do corpus. A ideia era testar se as submáquinas melhoravam o desempenho do Identificador Morfológico e qual o ganho obtido, caso isso ocorresse.

Corpus	70%	75%	80%	85%	90%
Acurácia	91,98%	92,01%	92,16%	92,31%	92,49%
Diferença	1,29%	1,28%	1,23%	1,29%	1,08%

Tabela 3. Resultados do Etiquetador com Tecnologia Adaptativa

Os resultados apresentados na Tab. 3 mostraram que o uso da tecnologia adaptativa proporcionou ganhos consistentes em todos os cenários de testes, sendo maiores quando o tamanho do corpus de treinamento foi menor. Como as submáquinas ainda não incorporaram todas as regras linguísticas previstas no modelo, e o tamanho do vocabulário usado para montagem dos autômatos foi relativamente pequeno, é possível que o desempenho do Identificador Morfológico ainda possa ser melhorado, com o objetivo de chegar ao nível dos etiquetadores do estado da arte da Língua Portuguesa que chegam a 97% de acurácia [18,19].

Também foi testado o cenário no qual o tamanho do corpus de treino é relativamente pequeno, e se a tecnologia adaptativa poderia trazer melhores resultados neste cenário. Para isso, foram realizados dois outros experimentos. No primeiro, o tamanho do corpus de treinamento foi treinado inicialmente com 40% do corpus CINTIL e testado com o restante. Em seguida, o tamanho do corpus de treinamento foi reduzido para 35%, 30%, 25% e 5%, ficando sempre o restante para testes. Os resultados são apresentados na Tab.4.

Corpus	40%	35%	30%	25%	5%
Acurácia	88,66%	88,04%	87,33%	86,35%	64,90%

Tabela 4. Resultados do Etiquetador sem Tecnologia Adaptativa

No segundo experimento, os testes foram repetidos com o uso das submáquinas adaptativas. Os resultados são apresentados na Tab. 5.

Corpus	40%	35%	30%	25%	5%
Acurácia	90,47%	89,99%	89,35%	88,55%	67,06%
Diferença	1,81%	1,95%	2,02%	2,20%	2,16%

Tabela 5. Resultados do Etiquetador com Tecnologia Adaptativa

Percebe-se, novamente, que o uso da tecnologia adaptativa melhorou o desempenho do etiquetamento, neste caso, gerando resultados melhores do que no cenário anterior, o que permite concluir que quando o tamanho do corpus é reduzido, a influência da tecnologia adaptativa é mais significativa. Um aspecto interessante é imaginar o que aconteceria se fossem utilizadas apenas a tecnologia adaptativa e as regras linguísticas, pois esta configuração poderia ser interessante no cenário em que não houvesse corpus de treinamento ou se o tamanho do corpus treinamento fosse insuficiente para o uso de técnicas estatísticas.

III. CONSIDERAÇÕES FINAIS

Este artigo apresentou uma revisão dos conceitos de Tecnologia Adaptativa e de Processamento da Linguagem Natural, e, em seguida, descreveu a arquitetura do reconhecedor gramatical Linguístico, com ênfase no módulo Identificador Morfológico. Em seguida, foram apresentados experimentos nos quais as submáquinas adaptativas de etiquetamento morfológico foram avaliadas, comparando seu desempenho ao do etiquetamento realizado sem o uso de tecnologia adaptativa. Os resultados permitiram concluir ganho consistente com o uso de tecnologia adaptativa e possibilidade de obtenção de melhorias, visto que nem todas as regras linguísticas foram implementadas nas submáquinas usadas nos experimentos. Os experimentos também geraram um desdobramento, no sentido

de avaliar o desempenho do Identificador Morfológico, usando apenas tecnologia adaptativa e regras linguísticas, configuração que pode ser útil quando não houver corpus de treinamento ou quando o tamanho do corpus treinamento for insuficiente para o uso de técnicas estatísticas.

REFERÊNCIAS

[1] http://www.pcs.usp.br/~lta/

[2] Taniwaki, C. Formalismos adaptativos na análise sintática de Linguagem Natural. Dissertação de Mestrado, EPUSP, São Paulo, 2001.

[3] Menezes, C. E. Um método para a construção de analisadores morfológicos, aplicado à língua portuguesa, baseado em autômatos adaptativos. Dissertação de Mestrado, Escola Politécnica da Universidade de São Paulo, 2000.

[4] Padovani, D. Uma proposta de autômato adaptativo para reconhecimento de anáforas pronominais segundo algoritmo de Mitkov. Workshop de Tecnologias Adaptativas – WTA 2009, 2009.

[5] Moraes, M. Alguns aspectos de tratamento sintático de dependência de contexto em linguagem natural empregando tecnologia adaptativa, Tese de Doutorado, Escola Politécnica da Universidade de São Paulo, 2006.

[6] Rich, E.; Knight, K. Inteligência Artificial, 2. Ed. São Paulo: Makron Books, 1993.

[7] Vieira, R.; Lima, V. Linguística computacional: princípios e aplicações. IX Escola de Informática da SBC-Sul, 2001.

[8] Fuchs, C.; Le Goffic, P. Les Linguistiques Contemporaines.

[9] M. G. V.Nunes et al. Introdução ao Processamento das Línguas Naturais. Notas didáticas do ICMC Nº 38, São Carlos, 88p, 1999. Paris, Hachette, 1992. 158p.

[10] Sardinha, T. B. A Língua Portuguesa no Computador. 295p. Mercado de Letras, 2005.

[11] Rocha, R.L.A. Tecnologia Adaptativa Aplicada ao Processamento Computacional de Língua Natural. Workshop de Tecnologias Adaptativas – WTA 2007, 2007.

[12] Contier, A., Padovani D., Neto J.J. O reconhecedor gramatical adaptativo Linguístico: experimentos e resultados comparativos. Workshop de Tecnologia Adaptativa (11: 2017: São Paulo) Memórias do WTA 2017. – São Paulo; EPUSP, 2017. 138.

[13] Basilio, M. Formação e Classes de Palavras no Português do Brasil. Ed.Contexto, 2004.

[14] http://www.nilc.icmc.usp.br/tep2/

[15] Forney, G.D. J. IEEE. Proceedings of the IEEE, Volume 61, Issue 3, 1973.

[16] Bird, Steven, Edward Loper and Ewan Klein (2009), Natural Language Processing with Python. O'Reilly Media Inc.

[17] Branco, António, Francisco Costa, João Silva, Sara Silveira, Sérgio Castro, Mariana Avelãs, Clara Pinto and João Graça, 2010, "Developing a Deep Linguistic Databank Supporting a Collection of Treebanks: the CINTIL DeepGramBank ", In Proceedings, LREC2010 - The 7th international conference on Language Resources and Evaluation, La Valleta, Malta, May 19-21, 2010.

[18] Branco, A., Silva, J.: Evaluating solutions for the rapid development of state-of-the-art POS taggers for Portuguese. In: Proceedings of the 4th Language Resources and Evaluation Conference (LREC). (2004) 507–510.

[19] Silva, J.: Shallow processing of Portuguese: From sentence chunking to nominal lemmatization. Master's thesis, University of Lisbon (2007) Published as Technical Report DI-FCUL-TR-07-16.

Ana Teresa Contier formou-se em Letras-Português pela Universidade de São Paulo (2001) e em publicidade pela PUC-SP (2002). Em 2007 obteve o título de mestre pela Poli-USP com a dissertação: "Um modelo de extração de propriedades de textos usando pensamento narrativo e paradigmático".

Djalma Padovani formou-se em administração de empresas pela Faculdade de Economia e Administração da Universidade de São Paulo, em 1987 e obteve o mestrado em engenharia de software pelo Instituto de Pesquisas Tecnológicas de São Paulo - IPT, em 2008. Trabalhou em diversas empresas nas áreas de desenvolvimento de software e tecnologia de informação e atualmente atua no Laboratório de Dados da Serasa Experían.

João José Neto graduado em Engenharia de Eletricidade (1971), mestrado em Engenharia Elétrica (1975) e doutorado em Engenharia Elétrica (1980), e livre-docência (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 Tecnologia Adaptativa 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.

Aprendizagem de Regras de Decisão Utilizando Técnicas Adaptativas: Experimentos Preliminares

F. K. O. Takatuzi e R. L. Stange

***Resumo*—Uma das formas de aprendizagem de máquina é a indução de regras a partir de exemplos, particularmente a extração de regras de classificação. A maioria dos trabalhos nesta área, se concentram na geração de regras na forma intermediária de árvores de decisão, para posteriormente converter cada caminho da árvore em uma regra. Este artigo propõe um algoritmo de geração de regras que utiliza as árvores de decisão adaptativas como estrutura intermediária. O objetivo é que tal algoritmo, seja um candidato credível para uso em problemas de aprendizagem incremental de regras de classificação. O algoritmo foi testado e comparado com algoritmos que utilizam diferentes abordagens, tais como aprendizagem estatística, aprendizagem de regras e indução de árvores de decisão. Os experimentos apresentados, sugerem que o algoritmo adaptativo para geração regras possui um resultado próximo a alguns algoritmos clássicos. Para um primeiro experimento, a eficiência do algoritmo foi satisfatória, visto que o método de generalização do algoritmo ainda precisa de ajustes, já que este utiliza apenas a frequência relativa dos atributos para escolha de atributos que compõem cada regra.**

***Index Terms*—Tecnologia adaptativa, aprendizagem incremental, indução de regras, classificação.**

I. Introdução

A Aprendizagem de Máquina é uma área da Inteligência Artificial que estuda a capacidade que os sistemas computacionais possuem de aprender e modificar seu comportamento, através da experiência adquirida a fim de melhorar seu desempenho em execuções posteriores [1]. Existem diversos métodos que são utilizados para a busca de soluções de problemas em aprendizagem de máquina, tais como o Aprendizado Bayesiano, as Redes Neurais Artificiais (RNA), os métodos de indução de árvores de decisão e a aprendizagem de regras de decisão. Os métodos de aprendizagem baseados em regras são partes integrantes de sistemas em inteligência artificial, porém, a adoção desses métodos em problemas de classificação ainda é modesta [2]. Uma visão geral dos métodos baseados em regras se concentra em classes de regras do tipo "*Se..então*". Para [3] o conjunto de regras do tipo "*Se..então*"é uma das formas mais expressivas e legíveis para representar hipóteses em problemas de aprendizagem. As regras de decisão são utilizadas para representar o conhecimento obtido a partir dos dados em aplicações diversas [4] [5] [6].

De forma geral, a maioria dos algoritmos são não incrementais, ou seja, não possuem a capacidade de incluir novos exemplos de como instâncias particulares do problema são resolvidas ao longo de sua execução. A necessidade de buscar soluções alternativas de aprendizagem incremental é de grande relevância [7] [8] [9], pois existem diversas aplicações em que novas informações vão surgindo ou se alterando ao longo do tempo, formando assim um cenário em que a utilização de algoritmos incrementais é mais conveniente [10]. Em soluções não incrementais, sempre que uma nova informação é disponibilizada, o processo de aprendizagem é repetido e a estrutura que representa o conhecimento é reconstruída. Em árvores de decisão por exemplo, a reconstrução da árvore implica em um custo computacional elevado [11]. Dessa forma, a utilização de métodos não incrementais se torna ineficiente, uma vez que as informações repassadas ao sistema de aprendizagem estão em constante modificação. Neste caso, torna-se necessário implementar processos dinâmicos baseados na aprendizagem incremental, de maneira que a base de conhecimento possa ser atualizada sem que todo o processo de aprendizagem necessite ser repetido.

Um conceito que tem sido adotado na solução de problemas com características dinâmicas é a adaptatividade. De acordo com Neto (2011) a adaptatividade refere-se a capacidade que os sistemas guiados por regras possuem de alterar em seu próprio comportamento, em função de seu comportamento atual e de estímulos de entrada.

Neste trabalho, é proposto um algoritmo para aprendizagem incremental de regras de decisão a partir de exemplos. O algoritmo foi capaz de gerar, a partir de um conjunto de exemplos de treinamento, uma árvore de decisão como resultado, onde cada caminho da árvore é representado por um cojunto de regras. Tal árvore, possui características adaptativas que permitem que sua estrutura interna seja alterada durante o processo de aprendizagem e classificação, possibilitando uma nova forma de aprendizagem incremental.

II. Trabalhos Relacionados

Desde [12], a tecnologia adaptativa têm sido utilizada como uma alternativa para resolução de problemas complexos, principalmente pela capacidade de expressão dos dipositivos adaptativos que, por meio da adição da adaptatividade, permite tornar dispositivos convencionais, tais como autômatos finitos e árvores de decisão, mais expressivos [13]. Outra característica atribuída aos dispositivos adaptativos é que, por serem baseados em dispositivos guiados por regras, podem proporcionar uma melhor interpretação pelos especialistas do domínio. Essa é uma das desvantagens de muitas técnicas de modelagem estatística, comumente usadas em aprendizado de máquina, o modelo resultante é difícil de interpretar [14]. Estudos preliminaries, apresentados em [15], mostram um caso particular da utilização da adaptatividade em aprendizagem, onde as tabelas de decisão adaptativas são utilizadas para representar o modelo de aprendizagem. Este trabalho mostra que

Os autores podem ser contatados através dos seguintes endereços de correio eletrônico: kenjitakatuzi@gmail.com, rlgomes@utfpr.edu.br.

este formalismo permite representar de maneira satisfatória um problema de aprendizagem de máquina, bem como possibilitar a identificação de melhorias para o processo de aprendizagem, pois o formato das regras facilita a interpretação do modelo de aprendizagem pelo projetista ou especialista.

[10], propuseram um algoritmo de indução de árvores de decisão utilizando técnicas adaptativas, que combina estratégias sintáticas e estatísticas, chamado *AdapTree*. Neste caso, não parece ser vantajoso considerar que os dispositivos adaptativos apenas melhoram a expressividade e compreensibilidade da solução, comparados a dispositivos não adaptativos. Assim, objetivo é melhorar o entendimento das regras, mas também cumprir exigências de eficiência.

Em [16] foi apresentado um método híbrido para aprender regras de classificação baseado na estratégia de cobertura sequencial, cujos métodos aplicam uma estratégia de decomposição do problema, na qual a tarefa de encontrar uma base de regras completa é reduzida a uma sequência de subproblemas, onde para cada subproblema a solução é uma única regra. Além disso, foi incorporado ao método a capacidade de automodificação do conjunto de regras, que permite inspecionar iterativamente o conhecimento extraído, podendo substituir uma ou mais regras para simplificar o conhecimento ou melhorar a capacidade de previsão do conjunto.

III. Referencial Teórico

A. Aprendizagem de regras de decisão

Uma visão geral dos métodos baseados em regras para representação do conhecimento e aprendizagem se concentra em classes de regras do tipo "*Se..então*". Para [3] o conjunto de regras deste tipo é um das formas mais expressivas e legíveis para representar hipóteses em aprendizagem de máquina. A forma geral de uma regra "*Se..então*"é:

`P` $\rightarrow$ `Q` ou `Se P então Q`, onde:

- `P` é uma proposição que pode conter um conjunto arbitrário de n pares de atributos - valor, $P = \text{condition}_1 \wedge \ldots \wedge \text{condition}_n$ (n é o tamanho da regra).
- `Q` é o valor do atributo categórico da classe.

Segundo [3], uma forma de aprender um conjunto de regras é construir uma árvore de decisão e converter o resultado para um conjunto de regras. A árvore de decisão pode ser mapeada para um conjunto de regras, transformando cada caminho da raiz até cada nó folha em uma regra. Outra maneira é extrair as regras diretamente do conjunto de treinamento, sem utilizar uma estrutura intermediária. Em resumo, as formas de aprender um conjunto de regras são:

- **Método 1:** Construir uma árvore de decisão e fazer a conversão para um conjunto de regras. Dado um conjunto de treinamento P, os passos para a contrução de uma árvore de decisão são [17]
  1) Se P contém um ou mais exemplos, todos pertencentes à mesma classe c_i então a árvore de decisão para P é um nó folha rotulado pela classe c_i;
  2) Se P não contém exemplos então a árvore é uma folha e a classe associada deve ser determinada a partir de informações além de P (ex: utilizar a classe mais frequente para o nó-pai desse nó);
  3) Se P contém exemplos que pertencem a várias classes então dividir P em subconjuntos mais "puros".
  4) Os passos 1, 2 e 3 são aplicados recursivamente para cada subconjunto de exemplos de treinamento de maneira que, em cada nó, as arestas levam para as subárvores construídas a partir do subconjunto de exemplos P.
- **Método 2:** Utilizar um algoritmo de cobertura sequencial (Sequential Covering):
  1) Invocar APRENDA-UMA-REGRA sobre todos os exemplos;
  2) Remover os exemplos cobertos pela regra aprendida;
  3) Repetir o processo até atingir a fração desejada de exemplos cobertos pelas regras.

B. Dispositivos Adaptativos

Um dispositivo guiado por regras é qualquer abstração formal que tem seu comportamento descrito por um conjunto finito de regras. No formalismo adaptativo, dispositivos adaptativos poder ser obtidos a partir da incorporação de um mecanismo adaptativo atrelado aos recursos oferecidos por um dispositivo guiado por regras. O mecanismo adaptativo é representado por um conjunto de funções adaptativas, as quais são conectadas às regras do dispositivo guiado por regras. As funções adaptativas são capazes de alterar o conjunto de regras do dispositivo, ocasionando a mudança de comportamento no dispositivo adaptativo.

Ações adaptativas podem ser definidas em termos de abstrações que são as funções adaptativas, de modo similar às chamadas de funções em linguagens de programação. As funções adaptativas são descritas por um cabeçalho e um corpo. O cabeçalho da função é composto por um nome, variáveis, parâmetros e geradores. Uma função adaptativa é referenciada por um nome ϕ, os parâmetros p são uma ênupla ordenada (ρ_1, ρ_2,..., ρ_n), de $n \geq 0$ parâmetros formais, as variáveis v são formadas por um conjunto (ν_1, ν_2,..., ν_m), de $m \geq 0$ variáveis ν, os geradores g formam um conjunto (γ_1, γ_2,..., γ_k), de $k \geq 0$ geradores. Os valores das variáveis são preenchidos uma única vez pelas ações elementares de inspeção da função adaptativa. Já os valores dos geradores são atualizados a cada chamada da função adaptativa. O corpo da função é composto por uma função adaptativa anterior e uma função adaptativa posterior e o núcleo da função.

As funções adaptativas, em seu núcleo, referem-se a operações básicas de edição do conjunto de regras que definem o dispositivo adaptativo, representadas pelas ações adaptativas elementares de inspeção, remoção e inserção de regra, onde:

1) *ação adaptativa elementar inspeção*: permite a consulta ao conjunto de regras em busca de um dado padrão de regra.
2) *ação adaptativa elementar remoção*: remove regras do conjunto corrente correspondentes a um dado padrão de regra.

3) *ação adaptativa elementar inserção*: insere uma regra no conjunto corrente de acordo com um dado padrão de regra.

A formulação para dispositivos guiados por regras e dispositivos adaptativos pode ser encontrada em [18].

C. Árvores de Decisão Adaptativas

Em [19], foi apresentado um dispositivo adaptativo cujo mecanismo subjacente é uma árvore de decisão. Esse dispositivo, chamado de árvore de decisão adaptativa, permite que a estrutura hierárquica de uma árvore de decisão possa ser dinamicamente alterada durante o processo de decisão, quando a árvore é percorrida da raiz para as folhas. A capacidade de automodificação é obtida através das funções adaptativas incorporadas a alguns ramos da árvore que permitem a inspeção, a inserção e remoção de subárvores. O mecanismo subjacente das árvores de decisão adaptativas é denominado árvores de decisão não-determinísticas e generaliza o conceito de árvore de decisão. A formulação da árvore de decisão adaptativa é apresentada a seguir, de acordo com [19].

Definição 1. Uma árvore de decisão não-determinística, ou árvore-DND, é uma 7-upla T =(I, Σ, Γ, f, c, A, R), onde:

- I: é o conjunto de exemplos;
- Σ: é o conjunto de atributos, incluindo o atributo classe;
- Γ: são os valores de atributos, incluindo o símbolo "?", denominado valor ausente que deve ser usado para representar valores ausentes, desconhecidos ou inexistentes;
- f: $\Sigma \rightarrow 2^{\Gamma}$: é uma função que determina o domínio de cada atributo;
- $c \in \Sigma$: define o atributo classe;
- A: $I \times \Sigma \rightarrow \Gamma$ é uma função binária usada para descrever os elementos do conjunto de exemplos, associando exemplos, atributos e valores de atributos.
- R: é uma estrutura hierárquica finita denominada subárvore e definida recursivamente como:
  - **Folha**: contendo um identificador único e um valor para o atributo classe;
  - **Não Folha**: uma (n+2-upla) (id, a, (v_1, R_1),..., (v_n, R_n)), onde id é um identificador, $a \in \Sigma$ -{c} é um atributo, $v_i \in f(a)$ são valores para $1 \leq i \leq n$ e todos os elementos R_i, $1 \leq i \leq n$ são subárvores.

A árvore de decisão não-determinística adaptativa, ou árvore-DND adaptativa, é uma árvore DND acrescida de uma camada adaptativa, que permite a modificação do conjunto R. Estas alterções podem ocorrer antes ou depois de cada execução de uma ação elementar, que podem inspecionar, remover ou inserir subárvores.

Definição 2. Uma árvore-DND adaptativa é uma dupla AT = (CS_0, CA), onde:

- CS_0: é o mecanismo subjacente, com uma leve modificação que permite o acoplamento de ações adaptativas a cada uma das subárvores de R;
- CA: mecanismo adaptativo, formado pelo conjunto das funções adaptativas.

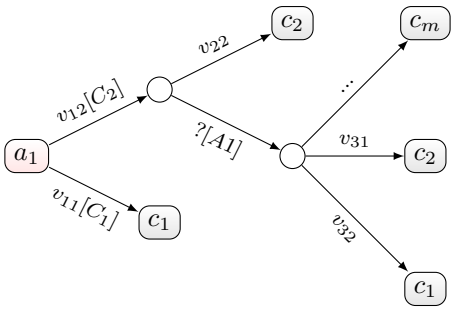


Figura 1. Exemplo de árvore de decisão adaptativa.

A operação de uma árvore- DND adaptativa opera de acordo com o mecanismo subjacente, até que uma função adaptativa seja disparada.

IV. Algoritmo de aprendizagem

O algoritmo de aprendizagem está baseado no funcionamento do *Adaptree*, proposto por [20], no qual a ideia geral é transformar cada exemplo de treinamento em um caminho partindo da raiz até a folha. Porém é necessário evidenciar algumas diferenças: 1) As funções adaptativas estão acopladas nos ramos e não aos nós como no *Adaptree*, isso permite a remoção e inserção de ramos e não de subárvores. 2) Cada caminho da árvore é transformado em uma regra do tipo "*Se..então*".

A árvore de decisão adaptativa gerada é composta pelo conjunto de atributos Σ, que possui uma ordenação arbitrária podendo ser representada por uma sequência $a_1, ..., a_n$, na qual n = número total de atributos, $c = a_n$ representa o atributo classe. A estrutura inicial S da árvore contém um valor inicial desconhecido ? na raiz e a camada adaptativa Φ é composta por um conjunto de funções adaptativas $A_1, ..., A_k$, podendo estas conter ações elementares de inserção ($+$), remoção ($-$) ou consulta (?). As funções adaptativas estão acopladas a alguns ramos específicos e quando acionadas, conferem à árvore a capacidade de automodificação.

O processo de aprendizagem pode ser divido em três etapas distintas, sendo elas *pré-processamento*, *aprendizagem* e *classificação*, que serão descritas a seguir:

A. Pré-processamento

O algoritmo realiza a leitura e pré-processamento do conjunto de dados, executando uma função inicial que impõe uma ordem aos atributos, de acordo com a forma que estão dispostos no conjunto de treinamento. Esta ordem é mantida ao longo de toda a execução do algoritmo. Por convenção, o primeiro atributo a ser lido é alocado na raiz da árvore, os demais atributos são os nós internos e o último atributo é a classe.

B. Aprendizagem

O algoritmo recebe uma nova instância de treinamento e busca na estrutura da árvore um caminho que corresponda

a instância que está sendo processada. Esta correspondência está baseada na comparação dos valores do atributo raiz e do atributo folha, ou seja, o algoritmo verifica se existe um caminho na estrutura S, cujo valor do atributo raiz e o valor do atributo folha sejam iguais aos valores da instância de treinamento. Na aprendizagem o algoritmo executa em dois modos: modo de aprendizagem não adaptativo e modo de aprendizagem adaptativo.

1) Modo de aprendizagem não adaptativo. Se não existir um caminho correspondente, o algoritmo constrói um novo caminho, de maneira que o valor do nó raiz e do nó folha assumem os valores contidos na instância de treinamento. Os valores dos nós internos são denotados por um valor "?", o qual representa um valor de atributo ainda desconhecido. Esses valores poderão ser preenchidos com base nos valores de atributos das futuras instâncias de treinamento. Além disso, aos ramos que possuem este símbolo, também está acoplada uma função adaptativa de classificação. Essas funções adaptativas poderão ser utilizadas posteriormente para a classificação de uma nova instância. Quando existe um caminho correspondente à instância de treinamento, o algoritmo executa o mecanismo estatístico que busca o atributo de maior relevancia, para modificar a árvore e melhorar a classificação. A medida estatística utilizada é baseada na frequência relativa de cada valor de atributo lido no caminho encontrado, em relação ao total de instâncias recebidas até o momento.

2) Modo de aprendizagem adaptativo. Se existir um caminho, o algoritmo por meio das funções adaptativas de aprendizagem (Algoritmo 1) acopladas aos ramos do nó raiz, modifica convenientemente este caminho de forma a melhor generalizar a árvore. A função adaptativa recebe como parâmetro o atributo-valor mais relevante, encontrado pelo mecanismo estatístico. Assim, por meio do mecanismo adaptativo é realizada a atualização do ramo correspondente ao atributo relevante.

Algoritmo 1: FUNÇÃO ADAPTATIVA DE APRENDIZAGEM.

1 **Função Adaptativa** A_k (a_j, a_{j+1}, $valor$)
2 **início**
3 | Consulta: $?[(a_j, ?) \rightarrow a_{j+1}]$
4 | Remoção: $-[(a_j, ?) \rightarrow a_{j+1}]$;
5 | Inserção: $+[(a_j, valor) \rightarrow a_{j+1}]$
6 **fim**

C. Classificação

O algoritmo recebe uma nova instância de teste a fim de classificá-la. Para isso, realiza uma busca em profundidade, percorrendo cada um dos caminhos da árvore até que se encontre um caminho capaz de classificar a nova instância. Nesta etapa, a utilização da tecnologia adaptativa torna-se possível conferir à árvore a capacidade de automodificação durante a classificação. Neste caso, se existir algum caminho da árvore com valores de atributo *default*, ou seja, contendo um valor "?", uma função adaptativa é disparada. Essa função executa uma ação elementar de remoção e outra de inserção (Algoritmo 2), que permite modificar o valor do atributo *default*, para um valor que permita a classificação da instância de teste. Os parâmetros a_j e *valor* são referentes à instância corrente. Após a classificação da instância, essa alteração é descartada e a árvore original é preservada para uma nova classificação.

Algoritmo 2: FUNÇÃO ADAPTATIVA DE CLASSIFICAÇÃO.

1 **Função Adaptativa** C_k (a_j, a_{j+1}, valor)
2 **início**
3 | Remoção: $-[(a_j, ?) \rightarrow a_{j+1}]$;
4 | Inserção: $+[(a_j, valor) \rightarrow a_{j+1}]$
5 **fim**

Por exemplo, supondo a execução da regra R_5, mostrada a seguir:

- Regra R5 antes da modificação:
  `Se` $((a_1 = v_{12}) \wedge (a_2 = v_{21}) \wedge (a_3 = ?)[C_1] \wedge (a_4 = v_{43}))$ `então` c_1

Para a classificação da instância $\alpha = v_{12}v_{21}v_{33}v_{43}$ a função adaptativa C_1 é acionada, tendo como parâmetro $a_j = a_3$ e $valor = v_{33}$. Neste caso, é executada a ação elementar de remoção $-[(a_3, ?) \rightarrow a_4]$ e a ação elementar de inserção $+[(a_3, a_3) \rightarrow a_4]$. A regra resultante da modificação é apresentada na sequência e classifica a instância como pertencente a classe c_1.

- Regra R5 depois da modificação:
  `Se` $((a_1 = v_{12}) \wedge (a_2 = v_{21}) \wedge (a_3 = v_{33}) \wedge (a_4 = v_{43}))$ `então` c_1

Quando a função de busca retorna mais de uma regra ou caminho, a decisão de classificação é tomada estatísticamente.

V. EXPERIMENTOS PRELIMINARES

O algoritmo foi implementado utilizando a linguagem de programação Java e reutiliza algumas classes do pacote WEKA (Waikato Environment for Knowledge Analysis) [21].

A. Recursos Utilizados

Dentre as limitações do algoritmo, está o fato dele trabalhar apenas com valores de atributos nominais e também ainda não realizar tratamento de valores ausentes (*missing value*), por este motivo foi utilizado apenas 1 conjunto de dados. O conjunto de dados utilizado foi o *lenses* contendo 24 instâncias de treinamento e 4 valores de atributos. Este conjunto de dados descreve quais tipos de lentes de contato se ajustam melhor a cada paciente. Os atributos observados são idade, astigmatismo e produção de lágrima. As classes indicam se o paciente deve usar lentes macias, lentes duras ou se não devem usar lentes.

Quanto aos algoritmos escolhidos, optou-se por utilizar algoritmos de diferentes abordagens de aprendizagem de máquina, por se tratar ainda de estudos preliminares. Os algoritmos utilizados foram:

- NaiveBayesUpdateable: versão incremental do Naive-Bayes;
- JRip ou Ripper: aprendizagem incremental de regras;
- HoeffdingTree: aprendizagem incremental de árvores de decisão;

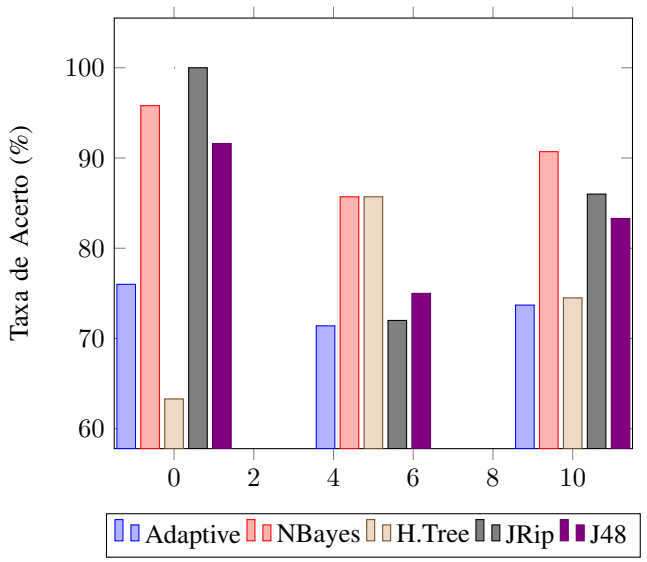


Figura 2. Gráfico comparativo das taxas de acerto

- J48: algoritmo não incremental da família TDIDT.

Os conjuntos de dados utilizados foram retirados do repositório do UCI (UC Irvine Machine Learning Repository) [1]. Todos os algoritmos utilizados possuem o código-fonte disponível na ferramenta WEKA [2].

A medida de desempenho utilizada foi a precisão, baseada nas quantidades de exemplos classificados corretamente, e na porcentagem de acertos obtidos pelo algoritmo.

Os testes foram realizados da seguinte forma:

- **Experimento 1**: foi utilizado o conjunto de dados completo tanto para treino quanto para teste.
- **Experimento 2**: o conjunto de dados completo foi dividido aleatoriamente em dois peças, um conjunto de treinamento e um conjunto de teste, na proporção aproximada de 70% a 30%.

B. Análise dos Resultados

A Tabela I apresenta as taxas de acerto para cada algoritmo, executados sobre cada um dos conjuntos de treinamento e testes. A primeira linha refere-se à Experimento 1, a linha dois refere-se à Experimento 2 e a última linha apresenta a média dos dois experimentos. A Figura 2 apresenta um gráfico dos resultados apresentados na Tabela I.

Tabela I
COMPARAÇÃO DAS TAXAS DE ACERTO.

Dados	***Adaptive***	**NBayes**	**H. Tree**	**JRip**	**J48**
Exp. 1	76%	95.8%	63.3	100%	91.6
Exp. 2	71.4%	85.7%	85.7%	72%	75%
Média	73.7%	90.7%	74.5%	86%	83.3%

De acordo com o gráfico, o resultado do novo algoritmo não oscilou muito entre os dois tipos de experimento, obtendo uma diferença de apenas 1%, o que pode ser considerado pouco significativo em termos estatísticos. Os algoritmos NaiveBayesUpdateable e HoeffdingTree obtiveram o maior desempenho no Experimento 2, porém este último obteve o pior desempenho no Experimento 1. No Experimento 1 o algoritmo JRip foi o algoritmo que obteve a melhor performance, porém no Experimento 2 o seu resultado foi um dos piores, com uma diferença inferior a 1% do Adaptive. Esse comportamento também foi observado no J48, que obteve um bom resultado no Experimento 1, e uma pequena queda no Experimento 2. Na média, os algoritmos que menos oscilaram os resultados foram primeiramente o Adaptive, em segundo o NaiveBayesUpdateable e na sequência o J48. É importante destacar que o JRip e o J48 são algortimos não incrementais, desta forma eles possuem a vantagem de possuir todo o conjunto de treinamento de antemão para a tomada de decisão e construção do classificador. Já em relação ao NaiveBayesUpdateable, apesar de ser incremental, este possui vantagem em relação ao Adaptative quanto ao método estatístico utilizado.

[1]Disponível em: `http://archive.ics.uci.edu/ml/index.php`
[2]Disponível em: `http://www.cs.waikato.ac.nz/ml/weka/`

VI. CONSIDERAÇÕES FINAIS

Para um estudo preliminar, os resultados foram considerados satisfatórios, visto que mesmo o algoritmo tendo obtido um desempenho inferior a outros algoritmos como o NaiveBayesUpdateable (versão incremental do NaiveBayes) e o J48 (implementação do C4.5), o resultado obtido em Experimento 2 foi similar ao JRip, um algoritmo incremental que também utiliza uma abordagem de aprendizagem de regras. A busca por melhores resultados pode ser conduzida de diferentes modos em trabalhos futuros. Primeiro, foi utilizado apenas um conjunto de dados, além disso nota-se que o conjunto de dados é relativamente pequeno, sendo uma extensão natural deste trabalho a coleta de novos dados. Porém, para isso será necessário implementação o tratamento de valores ausentes e um método de discretização para lidar com atributos numéricos. Em segundo, notou-se que a regras geradas pelo algoritmo ainda possuem muitos valores de atributos *default*, e isso dificulta a classificação, pois faz com que o algoritmo retorne vários caminhos possíveis para classificar uma dada instância e a decisão seja tomada pelo mecanismo estatítico. Neste ponto, considera-se a introdução de uma medida estatística mais sofisticada, como o ganho de informação por exemplo, já que a frequência relativa é uma medida bastante ingênua para a escolha do melhor atributo.

Além disso, em trabalhos futuros pretende-se realizar novos experimentos em função da complexidade das regras geradas pelo algoritmo. A complexidade de um conjunto de regras pode ser medida em termos de, pelo menos, três parâmetros [22]

Intuitivamente, o algoritmo parece ter bom desempenho em conjunto de dados onde pelo menos uns dos atributos é claramente mais relevante para uma determinada classe. Para conjunto de dados que em que todos os atributos contribuem em igual peso na resposta, a taxa de acerto do algoritmo é reduzida em função da medida estatística adotada. Porém para isso, serão necessários novos experimentos e comparação apenas com algoritmos baseados em regras.

REFERÊNCIAS

[1] E. Alpaydin, *Introduction to Machine Learning*, ser. Adaptive computation and machine learning. MIT Press, 2014. [Online]. Available: https://books.google.com.br/books?id=NP5bBAAAQBAJ

[2] R. Duda, P. Hart, and D. Stork, *Pattern Classification*. Wiley, 2012. [Online]. Available: https://books.google.com.br/books?id=Br33IRC3PkQC

[3] T. Mitchell, *Machine Learning*, ser. McGraw-Hill International Editions. McGraw-Hill, 1997. [Online]. Available: https://books.google.com.br/books?id=EoYBngEACAAJ

[4] D. Garcia, J. C. Gamez, A. Gonzalez, and R. Perez, "Using a sequential covering strategy for discovering fuzzy rules incrementally," in *2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, Aug 2015, pp. 1–8.

[5] M. Michalak, M. Sikora, and L. Wrobel, "Rule quality measures settings in a sequential covering rule induction algorithm - an empirical approach," in *2015 Federated Conference on Computer Science and Information Systems (FedCSIS)*, Sept 2015, pp. 109–118.

[6] F. E. B. Otero, A. A. Freitas, and C. G. Johnson, "A new sequential covering strategy for inducing classification rules with ant colony algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 1, pp. 64–76, Feb 2013.

[7] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang, "Incremental learning for robust visual tracking," *International Journal of Computer Vision*, vol. 77, no. 1, pp. 125–141, May 2008. [Online]. Available: https://doi.org/10.1007/s11263-007-0075-7

[8] R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments," *IEEE Transactions on Neural Networks*, vol. 22, no. 10, pp. 1517–1531, Oct 2011.

[9] S. Calinon and A. Billard, "Incremental learning of gestures by imitation in a humanoid robot," in *2007 2nd ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, March 2007, pp. 255–262.

[10] H. Pistori and J. J. Neto, "Adaptree - proposta de um algoritmo para indução de árvores de decisão baseado em técnicas adaptativas." in *Anais Conferência Latino Americana de Informática - CLEI 2002.*, Montevideo, Uruguai, Novembro 2002.

[11] M. L. Yoshida, "Aprendizado supervisionado incremental de redes bayesianas para mineração de dados." Master's thesis, Universidade Federal de São Carlos - UFSCar, 2007.

[12] J. J. Neto, *Solving Complex Problems Efficiently with Adaptive Automata*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 340–342.

[13] J. a. J. Rocha, Ricardo. L. A.; Neto, "Automato adaptativo, limites e complexidade em comparação com máquina de turing." in *Proceedings of the second Congress of Logic Applied to Technology - LAPTEC*. São Paulo: Faculdade SENAC de Ciências Exatas e Tecnologia, 2001, pp. 33–48.

[14] T. A. Plate, "Accuracy versus interpretability in flexible modeling: implementing a tradeoff using gaussian process models," *Behaviour Metrika Special Issue On Interpreting Neural Network Models*, vol. 26, pp. 29–50, 1999.

[15] R. L. Stange and J. J. Neto, "Aprendizagem incremental usando tabelas de decisão adaptativas." *Quinto Workshop de Tecnologia Adaptativa - WTA 2011*, 2011.

[16] ——, "Learning decision rules using adaptive technologies: a hybrid approach based on sequential covering," *Procedia Computer Science*, vol. 109, no. Supplement C, pp. 1188 – 1193, 2017, 8th International Conference on Ambient Systems, Networks and Technologies, ANT-2017 and the 7th International Conference on Sustainable Energy Information Technology, SEIT 2017, 16-19 May 2017, Madeira, Portugal. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050917310712

[17] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, Mar. 1986. [Online]. Available: http://dx.doi.org/10.1023/A:1022643204877

[18] J. J. Neto, *Adaptive Rule-Driven Devices - General Formulation and Case Study*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 234–250.

[19] H. Pistori, J. J. Neto, and M. C. Pereira, "Adaptive non-deterministic decision trees: General formulation and case study," *INFOCOMP Journal of Computer Science*, 2006.

[20] H. Pistori, "Tecnologia em engenharia de computação: estado da arte e aplicações," PhD thesis, Escola Politécnica, Universidade de São Paulo, 2003.

[21] I. Witten, E. Frank, and M. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, ser. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2011. [Online]. Available: https://books.google.com.br/books?id=bDtLM8CODsQC

[22] M. Bramer, *Automatic Induction of Classification Rules from Examples Using N-Prism*. London: Springer London, 2000, pp. 99–121.

Fabio Kenji Oshiro Takatuzi é graduado em Tecnologia em Sistemas para Internet pela Universidade Tecnológica Federal do Paraná (UTFPR) (2017). Atualmente, atua como instrutor de ensino na empresa VitalNet Consultoria em Informática, possuindo experiência no ensino de linguagens de programação, desenvolvimento web e design gráfico. Possui interesse na área da aprendizagem de máquina, em especial, na utilização da tecnologia adaptativa em aplicações para mineração de dados, tomada de decisão e inteligência artificial.

Renata Luiza Stange Atualmente é doutoranda 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 (PCS/EPUSP), atuando como pesquisadora no Laboratório de Linguagens e Técnicas Adaptativas (LTA). Mestre em Ciências, também pelo Programa de Pós-Graduação em Engenharia de Computação do PCS/EPUSP (2011). Bacharel em Análise de Sistemas pela Universidade Estadual do Centro-Oeste (2002). Atua como docente na Universidade Tecnológica Federal do Paraná (UTFPR). Tem experiência na área de Ciência da Computação, particularmente em dispositivos adaptativos, tecnologia adaptativa e suas aplicações à inteligência artificial, aprendizagem de máquina e reconhecimento de padrões.

Algoritmo de Earley Adaptativo para Reconhecimento de Padrões Sintáticos

G. P. Santos[1], J. J. Neto[2] e A. Machado-Lima[3]

***Resumo*—A teoria das linguagens formais é amplamente utilizada nos processos de solução de problemas de naturezas diversas, como por exemplo reconhecimento de padrões sintáticos. Gramáticas adaptativas, em particular, representam soluções eficientes quando se necessita incluir na modelagem dependências de contexto. Embora existam eficientes analisadores sintáticos para essas gramáticas, a grande maioria identifica apenas uma árvore sintática de uma dada cadeia ou exige que a gramática esteja em alguma forma normal. No entanto, há problemas para os quais se necessita obter todas as árvores de derivação e/ou utilizar gramáticas sem restrições aos formatos das produções. Como exemplo podemos citar a modelagem de famílias de RNAs com estrutura secundária. A fim de preencher essa lacuna, este artigo propõe uma versão adaptativa do algoritmo analisador sintático de Earley. Tal implementação fará parte de um arcabouço de geração de classificadores baseados em gramáticas que está sendo aprimorado com a incorporação de gramáticas adaptativas. A adaptatividade será uma importante ferramenta no tratamento do problema de caracterização de famílias de RNAs contendo pseudonós.**

***Palavras-chaves*—Reconhecimento de Padrões, Métodos Sintáticos, Métodos Adaptativos, Gramáticas, Earley**

I. Introdução

A teoria das linguagens formais, elaborada com o objetivo de desenvolver teorias relacionadas a linguagens naturais, logo passou a ser notada como importante também para o estudo das linguagens artificiais [1]. Assim, as linguagens formais passaram a ser utilizadas amplamente na análise sintática de linguagens de programação, na modelagem de circuitos e redes lógicas, em sistemas biológicos, sistemas de animação, hipertexto, linguagens não lineares e outros [1].

O conjunto de cadeias que compõem uma linguagem pode ser exaustivamente sintetizado por gramáticas, que são sistemas formais baseados em regras de substituição [2]. Para cada classe de linguagem há um tipo de gramática capaz de a representar, seguindo a hierarquia de Chomsky [3], que é organizada de acordo com a complexidade do formalismo.

As gramáticas podem ser utilizadas na área de reconhecimento de padrões sintáticos, uma vez que podem modelar a hierarquia dos componentes que formam as cadeias de uma linguagem, podendo ser traçado um paralelo entre esta hierarquia e a decomposição de padrões em subestruturas [4]. Assim, a árvore de derivação, ou árvore sintática, de uma gramática representa um padrão que pode ser utilizado para classificação.

Gramáticas podem também ser utilizadas no contexto de aprendizado estatístico, sendo neste caso utilizadas suas versões estocásticas. Uma gramática estocástica é uma gramática em que as regras de produção $P \subset \{\alpha \rightarrow \beta, p\}$, sendo $\alpha \in V^*NV^*$, $\beta \in V^*$, V o conjunto finito e não vazio de símbolos que representam o vocabulário da gramática e N o conjunto de símbolos não terminais da gramática, e p um valor de probabilidade, $0 \leq p \leq 1$, de tal forma a definir uma distribuição de probabilidades sobre as produções com o mesmo lado esquerdo, ou seja, considerando as produções $\alpha \rightarrow \beta_i, p_i \in P$, $\sum_i p_i = 1$.

As gramáticas estocásticas, ao invés de determinar se uma dada cadeia pertence à linguagem gerada pela gramática, atribui a ela uma probabilidade de pertencer à linguagem, ou seja, de ser gerada pela gramática. Essa probabilidade consiste no produto das probabilidades de todas as produções utilizadas na derivação da árvore sintática da cadeia. Se a gramática for ambígua, todas as árvores sintáticas da cadeia devem ser consideradas no cálculo desta probabilidade, somando-se as probabilidades dadas por cada árvore. Além disso, para gramáticas ambíguas, também é possível determinar qual a árvore sintática mais provável para uma dada cadeia, isto é, qual o padrão sintático mais provável associado à sua geração.

Considerando várias gramáticas estocásticas G_i, cada uma representando um padrão sintático, um classificador pode ser definido da seguinte forma: dada uma cadeia c, classifica-se a cadeia c como pertencente à linguagem que é gerada pela gramática G_K, sendo $K = \text{argmax}_i\ P(c|G_i)$.

A utilização de gramáticas estocásticas exemplifica a necessidade de algoritmos de análise sintática que sejam capazes de encontrar não só uma, mas todas as árvores sintáticas de uma cadeia quando a gramática em questão for ambígua.

A fim de facilitar a utilização de gramáticas em reconhecimento de padrões sintáticos, foi desenvolvido um arcabouço de geração classificadores baseados em gramáticas estocásticas, chamado GrammarLab.

Um dos problemas no quais o GrammarLab está sendo utilizado é o problema de Bioinformática de caracterização de famílias de RNAs. Tal caracterização, englobando tanto sequência quanto estrutura, exige gramáticas no mínimo livres de contexto. As regras de produção devem possuir um formato adequado para modelar os pareamentos existentes entre bases nucleotídicas do RNA, não podendo estar limitadas, por exemplo, à forma normal de Chomsky. Por isso o GrammarLab utiliza o analisador sintático de Earley [5], já que ele não assume que a gramática de entrada esteja em nenhuma forma normal e é capaz de encontrar todas as árvores sintáticas de uma cadeia. Mas para uma modelagem que inclua certos padrões estruturais conhecidos como pseudonós, é necessária a inclusão de dependência de contexto. Os pseudonós, por apresentarem relações de dependências cruzadas (Figura 1), tornam necessário o uso de gramáticas sensíveis ao contexto [6]. No

[1]EACH - Universidade de São Paulo - gilmar.santos@usp.br
[2]POLI - Universidade de São Paulo - joao.jose@poli.usp.br
[3]EACH - Universidade de São Paulo - ariane.machado@usp.br

entanto, o GrammarLab não possui suporte para gramáticas sensíveis ao contexto devido à alta complexidade de tempo da análise sintática dessa classe de gramáticas, cujo problema geral é NP-completo [7] [8] [9].

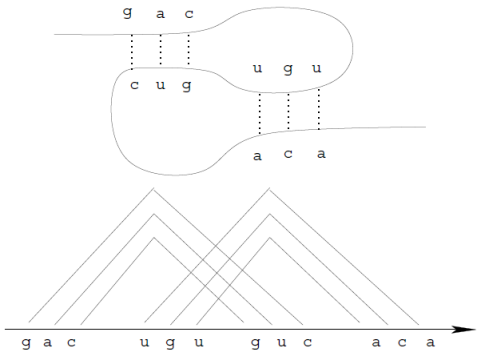


Figura 1. Estrutura e dependências cruzadas de um pseudonó.

A fim de sanar essa lacuna, este artigo apresenta uma proposta de evolução do algoritmo de Earley com a inclusão de adaptatividade em sua constituição, de tal forma que suas características como analisador sintático de gramáticas livres de contexto sejam expandidas para reconhecer gramáticas adaptativas, que por sua vez são capazes de modelar trechos dependentes de contexto sem aumentar excessivamente o tempo de análise.

Este é o primeiro passo de uma série no aprimoramento do GrammarLab, sendo que a partir desta melhoria o arcabouço será capaz de lidar com o problema de classificação de sequências de RNAs mesmo de famílias cujas estruturas apresentem elementos que necessitem de uma gramática no mínimo sensível ao contexto, sem no entanto cair em um problema NP-completo.

O restante deste artigo está organizado da seguinte forma. A seção II traz os conceitos básicos para o entendimento deste trabalho, a saber sobre analisadores sintáticos para gramáticas livres de contexto, o algoritmo analisador de Earley e dispositivos adaptativos. A seção III apresenta brevemente a atual versão do GrammarLab. A seção IV apresenta a proposta do algoritmo de Earley adaptativo. Finalmente, a seção V traz as conclusões deste trabalho.

II. Conceitos fundamentais

A. Analisadores Sintáticos

Para cada gramática livre de contexto pode ser desenhado um autômato a pilha que reconheça a linguagem gerada por tal gramática [10], sendo a complexidade de reconhecimento de ordem polinomial. Dada sua complexidade de reconhecimento e a capacidade de gerar todas as linguagens regulares e muitas linguagens adicionais [10], as gramáticas livres de contexto acabam se tornando um interessante objeto de estudo.

Outra alternativa de reconhecimento das linguagens livres de contexto são os algoritmos analisadores sintáticos de gramáticas, que diferem dos autômatos a pilha por serem de propósito geral, isto é, não serem desenhados para operar em uma linguagem livre de contexto específica. Dentre os algoritmos analisadores de gramáticas livres de contexto, podemos destacar o algoritmo CYK [11] e o algoritmo de Earley [5].

O algoritmo CYK utiliza uma estratégia de análise de baixo para cima (*bottom-up parser*) e programação dinâmica, resolvendo o problema de reconhecimento e determinação das diferentes árvores sintáticas com uma complexidade de tempo $O(n^3)$, sendo n o tamanho da sequência testada. Uma desvantagem do algoritmo CYK é a necessidade da gramática estar representada na forma normal de Chomsky. Uma gramática está na forma normal de Chomsky quando todas as suas regras de produção são da forma:

- $A \rightarrow BC, A \in N, B \in N, C \in N$
- $A \rightarrow \alpha, A \in N, \alpha \in \Sigma$

sendo N o conjunto de símbolos não terminais da gramática e Σ o conjunto finito e não vazio de símbolos que representa o alfabeto, os símbolos terminais, da gramática.

Toda gramática pode ser convertida para a forma normal de Chomsky [10]. No entanto, essa normalização pode afetar a informação estrutural fornecida pelas árvores sintáticas da gramática original.

B. Algoritmo de Earley

Diferentemente do algoritmo CYK, o algoritmo de Earley não necessita que a gramática esteja em uma forma específica, eliminando a necessidade de adaptação da gramática ou processamento adicional para normalizar uma gramática que se deseja analisar [5]. Além disso, o algoritmo de Earley [5] apresenta a característica de identificar as diferentes árvores de derivação, possuindo uma complexidade computacional de ordem $O(n^3)$ para gramáticas ambíguas. Isso faz com que seja um algoritmo conveniente, por exemplo, para o uso em problemas de bioinformática relacionados com análise de sequências biológicas, que por natureza são linguagens ambíguas, e que demandam que o formato das regras de produção mimetizem a estrutura de pareamentos entre as bases nucleotídicas (símbolos terminais na gramática).

Basicamente, o algoritmo de Earley percorre uma dada cadeia de entrada $X_1...X_n$ da esquerda para a direita. Para cada símbolo X_i percorrido, um conjunto de estados S_i é construído, representando a condição do processo de reconhecimento no ponto atual. Cada estado $s \in S_i$ é um ítem de análise que é representado por uma quádrupla, a qual é formada por: (i) uma produção da gramática; (ii) uma posição que indica até que ponto desta regra o reconhecimento foi realizado com sucesso; (iii) um ponteiro para a posição na cadeia de entrada em que se iniciou a busca pela instância da produção e (iv) uma cadeia formada pelos próximos k símbolos terminais seguintes à produção (geralmente é considerado apenas um).

Cada estado s é utilizado no desenvolvimento da análise sintática por meio da aplicação de uma dentre três possíveis operações do algoritmo: (i) *predictor*, que é a operação na qual são expandidos os símbolos não terminais das regras de produção, adicionando novos estados no conjunto S_i atual; (ii) *scanner*, operação na qual é verificado se o símbolo terminal sendo analisado da regra de produção do estado atual é o

símbolo X_i atual, gerando um novo conjunto de estados S_{i+1} e seguindo para o próximo símbolo de entrada X_{i+1} e (iii) *completer*, operação aplicada quando a produção do estado s foi analisada completamente, sendo comparados os k símbolos candidatos sucessores aos símbolos de entrada $X_{i+1}..X_{i+k}$, adicionando ao conjunto S_i atual os estados com as produções nas quais o símbolo completado aparece do lado direito.

Ao finalizar a varredura da cadeia de entrada, o algoritmo identifica se a cadeia pertence ou não à linguagem descrita pela gramática em questão e, caso afirmativo, pode retornar todas as árvores de derivação (utilizando os ponteiros dos estados para navegar pelo resultado da análise sintática).

C. Dispositivos Adaptativos

O uso de métodos adaptativos [2] possibilita alterar dinamicamente as propriedades de uma gramática, viabilizando inserir sensibilidade ao contexto em uma gramática originalmente livre de contexto sem, com isso, aumentar excessivamente sua complexidade de análise. Isso é possível pois as regiões dependentes de contexto de uma cadeia podem ser confinadas em regiões controladas. Ou seja, é realizada uma separação da cadeia em regiões localmente regulares ou livres de contexto, regiões estas posteriormente integradas em uma análise global dependente de contexto. Tal separação de análise pode resultar em uma economia considerável de tempo de reconhecimento.

Dispositivos adaptativos são dispositivos formais que podem ter seu comportamento alterado de forma dinâmica como resposta espontânea a estímulos de entrada [12].

Quaisquer alterações possíveis no comportamento de um dispositivo adaptativo devem ser conhecidas *a priori*. Assim, esses dispositivos são capazes de detectar as situações que disparam as modificações e devem ser automodificáveis para reagir de forma adequada, se adaptando à situação.

Um dispositivo adaptativo é formado pela incorporação de ações adaptativas às regras de um dispositivo não adaptativo subjacente. Assim, sempre que alguma dessas regras é aplicada, a ação adaptativa correspondente é acionada. Dessa forma, o dispositivo adaptativo resultante pode ser facilmente compreendido por todos que tenham familiaridade com o dispositivo subjacente.

Em [13] é apresentado um formalismo para uma gramática adaptativa sensível ao contexto e é estabelecida uma equivalência com autômatos adaptativos [14].

D. Gramática Adaptativa

Uma gramática adaptativa é um dispositivo adaptativo, conforme descrito em [13]. Nesta pesquisa, a gramática utilizada será representada formalmente por $G_A = (G^0, T, R^0)$, na qual:

- G^0 é a gramática livre de contexto inicial;
- T é um conjunto finito, possivelmente vazio, de funções adaptativas, que são conjuntos de ações adaptativas [12];
- R^0 é a relação entre as regras de produção da gramática G^0 e as funções adaptativas, sendo $R^0 \subseteq BA \times P^0 \times AA$;
- $BA \subseteq (T \cup \{\epsilon\})$ é o conjunto de ações adaptativas executadas antes do acionamento da respectiva regra de produção $p \in P^0$;
- $AA \subseteq (T \cup \{\epsilon\})$ é o conjunto de ações adaptativas executadas após o acionamento da respectiva regra de produção $p \in P^0$.

sendo P^0 o conjunto de produções da gramática G^0 e $\{\epsilon\}$ o conjunto que tem como único elemento ϵ, que representa uma ação adaptativa nula.

A cada ação adaptativa executada, uma nova gramática livre de contexto G^i é gerada.

Neste trabalho será considerado como dispositivo subjacente uma gramática livre de contexto, de forma a reduzir, no geral, a complexidade de análise.

Exemplo de Gramática Adaptativa

O seguinte exemplo, utilizando formalismo adaptativo [13] [12], apresenta uma gramática simplificada que representa a linguagem $L = a^n b^m c^n d^m$, que representa uma dependência cruzada na qual há uma relação entre os símbolos terminais a e c e entre os símbolos b e d.

$$
\begin{array}{l}
G_A = (G^0, T, R^0) \\
G^0 = \{V^0, \Sigma, P^0, S\} \\
V^0 = \{S, K, a, b, c, d\} \\
\Sigma = \{a, b, c, d\} \\
\\
P^0 = \{ \\
\quad S \rightarrow K\{AdP(K)\} \\
\quad K \rightarrow \phi \\
\} \\
\\
T = \{ \\
\quad AdP(X) = \{A'*, B'*, C'*, D'* : \\
\qquad +[X \rightarrow A'B'C'D'] \\
\qquad +[A' \rightarrow aA'\{Ad1(A', C', a, c)\}] \\
\qquad +[A' \rightarrow a\{Ad2(C', c)\}] \\
\qquad +[B' \rightarrow bB'\{Ad1(B', D', b, d)\}] \\
\qquad +[B' \rightarrow b\{Ad2(D', d)\}] \\
\qquad +[C' \rightarrow \phi] \\
\qquad +[D' \rightarrow \phi] \\
\quad \} \\
\\
\quad Ad1(N, X, x, y) = \{X'* : \\
\qquad -[N \rightarrow xN\{Ad1(N, X, x, y)\}\}] \\
\qquad -[N \rightarrow x\{Ad2(X, y)\}] \\
\qquad +[N \rightarrow xN\{Ad1(N, X', x, y)\}\}] \\
\qquad +[N \rightarrow x\{Ad2(X', y)\}] \\
\qquad +[X \rightarrow yX'] \\
\qquad +[X' \rightarrow \phi] \\
\quad \} \\
\\
\quad Ad2(X, y) = \{ \\
\qquad +[X \rightarrow y] \\
\quad \} \\
\}
\end{array}
$$

Neste exemplo, a regra de produção inicial $S \rightarrow K$ possui uma associação com uma função adaptativa $\{AdP(K)\}$, sendo

que a produção $K \rightarrow \phi$ indica que o símbolo não terminal K será definido de forma dinâmica, pelo acionamento de alguma ação adaptativa. A função adaptativa $AdP(X)$, ao ser acionada, basicamente indica o início de um domínio de dependência cruzada desmembrado em quatro partes, sendo A^i relacionado com C^i e B^i relacionado com D^i. Cada vez que uma regra de produção associada com A^i ou B^i é aplicada, uma nova regra associada a C^i e/ou D^i é adicionada ao conjunto de regras de produção, forçando a dependência cruzada.

Abaixo, uma derivação de uma dependência cruzada utilizando a gramática adaptativa de exemplo, e na Figura 2 o destaque para a dependência apresentada.

$$
\begin{aligned}
S &\Rightarrow_{G^0} K \\
&\Rightarrow_{G^1} A^1B^1C^1D^1 \\
&\Rightarrow_{G^1} aA^1B^1C^1D^1 \\
&\Rightarrow_{G^2} aaB^1C^1D^1 \\
&\Rightarrow_{G^3} aabB^1C^1D^1 \\
&\Rightarrow_{G^4} aabbB^1C^1D^1 \\
&\Rightarrow_{G^5} aabbbC^1D^1 \\
&\Rightarrow_{G^6} aabbbcC^2D^1 \\
&\Rightarrow_{G^6} aabbbccD^1 \\
&\Rightarrow_{G^6} aabbbccdD^2 \\
&\Rightarrow_{G^6} aabbbccddD^3 \\
&\Rightarrow_{G^6} aabbbccddd
\end{aligned}
$$

$$\Rightarrow_{G^6} aabbbccddd$$

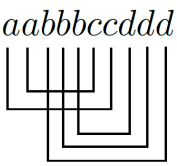


Figura 2. Dependência cruzada.

III. GrammarLab: Laboratório de Geração de Classificadores

O GrammarLab [15] é um arcabouço desenvolvido em C++ com o objetivo de facilitar a implementação e geração de classificadores baseados em gramáticas estocásticas, facilitando a implementação de algoritmos de inferência gramatical, de estimação das probabilidades para tornar uma gramática estocástica, de geração de analisadores sintáticos e de combinação dos mesmos para gerar os classificadores. Na Figura 3 é apresentada uma visão geral do arcabouço sendo utilizado para classificação de padrões sintáticos.

O arcabouço é composto por três partes: (i) algoritmos de inferência gramatical e estimação de probabilidades, (ii) módulo de suporte a implementação e (iii) módulo de suporte a testes.

A parte (i) é constituída de classes abstratas e concretas de inferidores gramaticais e de estimadores de probabilidades.

A parte (ii) é constituída de uma biblioteca de classes que implementam estruturas de dados úteis nesse contexto, como gramáticas, autômatos a árvore, trie e conjuntos de classes que modelam *streams* de entrada e saída que possibilitam um

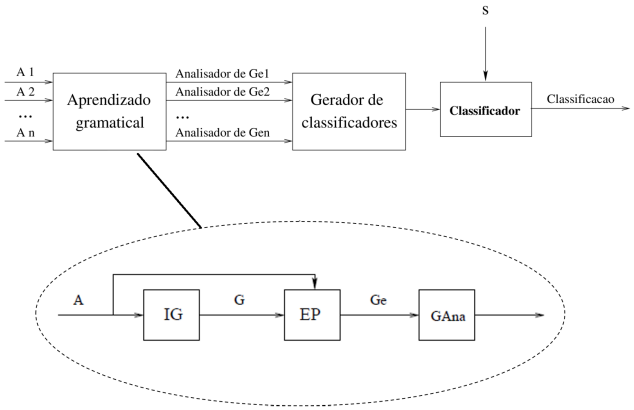


Figura 3. GrammarLab: visão geral.

canal de comunicação entre os algoritmos. Também são encontradas classes para geração de analisadores sintáticos e um mecanismo integrador de todos os analisadores das gramáticas consideradas na classificação. O algoritmo de Earley [5] foi escolhido como implementação para o analisador sintático de gramáticas livres de contexto estocásticas no arcabouço pelo fato de fornecer todas as árvores de derivação e por não exigir nenhuma normalização específica da gramática, conforme dito anteriormente.

A parte (iii) é formada por programas Perl e C++ que tratam a geração dos classificadores, execução de testes e obtenção de resultados.

Como abordagem para otimização de tempo de execução, o arcabouço utiliza uma estratégia de geração de códigos fontes (extensões .cpp e .h) para cada gramática inferida pelos algoritmos de aprendizado e seus analisadores sintáticos. Em contrapartida, tanto a gramática, quando os analisadores têm um comportamento totalmente estático.

IV. Proposta: GrammarLab Adaptativo

Uma vez que o arcabouço GrammarLab possui toda uma infraestrutura que atende às principais demandas para reconhecimento de padrões sintáticos para o problema alvo citado de caracterização de famílias de RNAs, como a implementação do algoritmo de Earley e várias classes de apoio para a implementação de classificadores, a proposta desse projeto é a evolução do arcabouço para que o mesmo possa lidar com gramáticas sensíveis ao contexto. Para isso, conforme visto na seção II-C, serão utilizados dispositivos adaptativos, de forma que a complexidade de análise sintática não seja aumentada excessivamente ao ponto de inviabilizar seu uso na prática.

A modelagem atual de classes C++ que representam uma gramática será alterada para permitir a inclusão de ações adaptativas associadas às regras de produção de uma gramática livre de contexto, seguindo o formalismo descrito em [12] e inspirado em [13].

O foco da primeira parte da pesquisa é a implementação da análise sintática baseada no algoritmo de Earley para gramáticas adaptativas, permitindo assim a classificação de sequências de estruturas que apresentam elementos que necessitem de uma gramática no mínimo sensível ao contexto, a exemplo dos

pseudonós encontrados nas estruturas secundárias de alguns RNAs.

O comportamento estático atual do GrammarLab precisará ser alterado para que possa comportar a alteração dinâmica nas regras de produções da gramática, comportamento característico dos dispositivos adaptativos.

A. Algoritmo de Earley Adaptativo

O algoritmo de Earley pode ser visto como um mapeamento de uma gramática livre de contexto em um conjunto de regras de análise R. Este conjunto de regras, por sua vez, pode ser considerado como descritor do dispositivo subjacente da adaptatividade.

Considerando que uma ação adaptativa A associada a uma produção P de uma gramática é executada sempre que P for ativada, temos que A é executada sempre que no conjunto R for ativado o grupo $R(P)$, ou seja, o conjunto de regras associado à ativação da produção P.

Se, ao ser especificada a gramática adaptativa proposta, cada uma de suas produções livres de contexto P for associada à respectiva ação adaptativa A, então será possível construir um a um os respectivos conjuntos de regras de análise $R(P)$ e, nessa mesma ocasião, poderá ser automaticamente acoplada a respectiva ação adaptativa A ao correspondente conjunto $R(P)$, de modo que, em tempo de execução, caso a produção P seja escolhida para ser aplicada, a ativação de A seja uma decorrência da ativação de $R(P)$, de forma que $P \rightarrow \{Apply(R(P)); Exec(A)\}$.

Assim, é necessária a alteração da operação *predictor* do algoritmo de Earley (descrito na seção II-B) para que seja aplicada a ação adaptativa e gerada uma nova gramática livre de contexto G^i, assim como a inclusão de um ponteiro para a gramática referenciada no controle de estados do algoritmo.

B. Exemplo de Aplicação do Algoritmo

A seguir, um exemplo de uma análise sintática considerando a gramática adaptativa e a cadeia de exemplo apresentadas na seção II-D.

Inicialmente, a cadeia de entrada é complementada com $k+1$ símbolos $\dashv$ não pertencentes ao alfabeto da gramática.

$$X = aabbbccddd \dashv\dashv$$

Passo $i = 0$, conjunto de estados S_0 e $X_1 = a$

Como passo inicial $i = 0$, o estado inicial é adicionado ao conjunto de estados S_0.

$$\phi \rightarrow .S \dashv \quad \dashv \;\; 0 \;\; G^0 \tag{1}$$

Nessa representação de um estado, $\phi \rightarrow S \dashv$ é a produção, o . indica até que ponto o reconhecimento já foi realizado com sucesso, o símbolo $\dashv$ representa a cadeia de símbolos terminais seguintes à produção, 0 é o ponteiro para a posição na cadeia de entrada em que se iniciou a busca pela instância da produção e G^0 representa a gramática resultante do acionamento da produção.

Como o símbolo após . é o símbolo não terminal S, a operação *predictor* é aplicada à produção inicial original da gramática G^0, ativando a função adaptativa associada com tal produção e gerando uma nova gramática G^1.

$$S \rightarrow .K\{AdP(K)\} \quad \dashv \;\; 0 \;\; G^1 \tag{2}$$

Novamente a operação *predictor* é aplicada ao novo estado, adicionando um novo estado no conjunto S_0, utilizando desta vez a produção da gramática G^1. Como não há uma função adaptativa associada com a produção, desta vez a gramática não evolui.

$$K \rightarrow .A^1B^1C^1D^1 \quad \dashv \;\; 0 \;\; G^1 \tag{3}$$

A operação *predictor* é aplicada mais uma vez. Como o símbolo seguinte a A^1 é o não terminal B^1, o mesmo é avaliado e são considerados os primeiros símbolos não terminais possíveis como a cadeia seguinte à produção, adicionando então os seguintes estados ao conjunto S_0.

$$A^1 \rightarrow .aA^1\{Ad1(A^1, C^1, a, c)\} \quad b \;\; 0 \;\; G^{2.1} \tag{4}$$

$$A^1 \rightarrow .a\{Ad2(C^1, c)\} \quad b \;\; 0 \;\; G^{2.2} \tag{5}$$

A operação *predictor* não pode ser aplicada a nenhum dos últimos estados adicionados, pois em todos eles o símbolo à direita do ponto é um símbolo terminal. Neste caso, a operação *scanner* é aplicada, adicionando ao conjunto de estados S_{i+1} os estados em que o símbolo terminal seguinte ao ponto for igual ao símbolo $X_{i+1} = a$ da cadeia de entrada. Os estados são copiados, tendo como alteração apenas a posição do ponto, que é deslocado um símbolo para a direita, indicando que o símbolo terminal anterior foi processado.

Assim, o conjunto de estados S_1, após a aplicação da operação *scanner* em todos os últimos estados de S_0, possui os seguintes estados:

$$A^1 \rightarrow a.A^1\{Ad1(A^1, C^1, a, c)\} \quad b \;\; 0 \;\; G^{2.1} \tag{6}$$

$$A^1 \rightarrow a\{Ad2(C^1, c)\}. \quad b \;\; 0 \;\; G^{2.2} \tag{7}$$

Passo $i = 1$, conjunto de estados S_1 e $X_2 = a$

Como não existem mais estados a serem processados no conjunto S_0, o algoritmo evolui para o passo $i = 1$ e o conjunto S_1 se torna o conjunto a ser processado.

Como o símbolo não terminal A^1 está após o ponto no estado 6, a operação *predictor* é aplicada, adicionando ao conjunto S_1 os estados:

$$A^1 \rightarrow .aA^1\{Ad1(A^1, C^2, a, c)\} \quad b \;\; 1 \;\; G^{3.1} \tag{8}$$

$$A^1 \rightarrow .a\{Ad2(C^2, c)\} \quad b \;\; 1 \;\; G^{3.2} \tag{9}$$

A operação *completer* é aplicada nos estados em que o ponto se encontra no fim da produção, sendo comparados os k símbolos da cadeia posterior do estado com os símbolos $X_{i+1}..X_{i+k}$ da cadeia de entrada. O estado 7 possui uma cadeia posterior divergente da cadeia de entrada, sendo portanto descartado.

Após a aplicação da operação *scanner* nos últimos estados de S_1, os seguintes estados são adicionados ao conjunto de estados S_2:

$$A^1 \rightarrow a.A^1\{Ad1(A^1, C^2, a, c)\} \quad b\ 1\ G^{3.1} \tag{10}$$

$$A^1 \rightarrow a\{Ad2(C^2, c)\}. \quad b\ 1\ G^{3.2} \tag{11}$$

O algoritmo evolui para o passo $i = 2$ na sequência.

Passo $i = 2$, conjunto de estados S_2 e $X_2 = b$

A operação *predictor* é aplicada ao estado 10, adicionando ao conjunto S_2 os estados:

$$A^1 \rightarrow .aA^1\{Ad1(A^1, C^3, a, c)\} \quad b\ 2\ G^{4.1} \tag{12}$$

$$A^1 \rightarrow .a\{Ad2(C^3, c)\} \quad b\ 2\ G^{4.2} \tag{13}$$

A operação *completer* é aplicada ao estado 11 e, como a cadeia posterior da produção é igual à cadeia de entrada $X_3 = b$ sendo processada, o conjunto de estados indicado pelo ponteiro no estado sendo analisado é percorrido, sendo adicionados ao conjunto de estados atual S_i todos os estados em que o símbolo à direita do ponto seja o símbolo não terminal do lado esquerdo da produção do estado em que a operação *completer* foi aplicada, sendo o ponto movido para a direita, indicando que o símbolo anterior foi processado. A gramática do estado em que a operação foi aplicada é mantida nos novos estados. O seguinte estado é então adicionado ao conjunto S_2:

$$A^1 \rightarrow aA^1\{Ad1(A^1, C^1, a, c)\} \quad b\ 0\ G^{3.2} \tag{14}$$

Os estados 12 e 13 são descartados pela operação *scanner*.

A operação *completer* é aplicada no estado 14, adicionando o novo estado:

$$K \rightarrow A^1.B^1C^1D^1 \quad \dashv\ 0\ G^{3.2} \tag{15}$$

A operação *predictor* é aplicada na sequência, adicionado os seguintes estados a S_2:

$$B^1 \rightarrow .bB^1\{Ad1(B^1, D^1, b, d)\} \quad c\ 2\ G^{4.3} \tag{16}$$

$$B^1 \rightarrow .b\{Ad2(D^1, d)\} \quad c\ 2\ G^{4.4} \tag{17}$$

As três operações continuam sendo aplicadas estado a estado, sendo os conjuntos de estados de cada passo do algoritmo apresentados resumidamente na sequência:

Passo $i = 3$, conjunto de estados S_3 e $X_4 = b$

$$B^1 \rightarrow b.B^1\{Ad1(B^1, D^1, b, d)\} \quad c\ 2\ G^{4.3} \tag{18}$$

$$B^1 \rightarrow b\{Ad2(D^1, d)\}. \quad c\ 2\ G^{4.4} \tag{19}$$

$$B^1 \rightarrow .bB^1\{Ad1(B^1, D^2, b, d)\} \quad c\ 3\ G^{5.1} \tag{20}$$

$$B^1 \rightarrow .b\{Ad2(D^2, d)\} \quad c\ 3\ G^{5.2} \tag{21}$$

Passo $i = 4$, conjunto de estados S_4 e $X_5 = b$

$$B^1 \rightarrow b.B^1\{Ad1(B^1, D^2, b, d)\} \quad c\ 3\ G^{5.1} \tag{22}$$

$$B^1 \rightarrow b\{Ad2(D^2, d)\}. \quad c\ 3\ G^{5.2} \tag{23}$$

$$B^1 \rightarrow .bB^1\{Ad1(B^1, D^3, b, d)\} \quad c\ 4\ G^{6.1} \tag{24}$$

$$B^1 \rightarrow .b\{Ad2(D^3, d)\} \quad c\ 4\ G^{6.2} \tag{25}$$

Passo $i = 5$, conjunto de estados S_5 e $X_6 = c$

$$B^1 \rightarrow b.B^1\{Ad1(B^1, D^3, b, d)\} \quad c\ 4\ G^{6.1} \tag{26}$$

$$B^1 \rightarrow b\{Ad2(D^3, d)\}. \quad c\ 4\ G^{6.2} \tag{27}$$

$$B^1 \rightarrow .bB^1\{Ad1(B^1, D^4, b, d)\} \quad c\ 5\ G^{7.1} \tag{28}$$

$$B^1 \rightarrow .b\{Ad2(D^4, d)\} \quad c\ 5\ G^{7.2} \tag{29}$$

$$B^1 \rightarrow bB^1\{Ad1(B^1, D^2, b, d)\}. \quad c\ 3\ G^{6.2} \tag{30}$$

$$B^1 \rightarrow bB^1\{Ad1(B^1, D^1, b, d)\}. \quad c\ 2\ G^{6.2} \tag{31}$$

$$K \rightarrow A^1B^1.C^1D^1 \quad \dashv\ 0\ G^{6.2} \tag{32}$$

$$C^1 \rightarrow .cC^2 \quad d\ 5\ G^{6.2} \tag{33}$$

Passo $i = 6$, conjunto de estados S_6 e $X_7 = c$

$$C^1 \rightarrow c.C^2 \quad d\ 5\ G^{6.2} \tag{34}$$

$$C^2 \rightarrow .c \quad d\ 6\ G^{6.2} \tag{35}$$

Passo $i = 7$, conjunto de estados S_7 e $X_8 = d$

$$C^2 \rightarrow c \quad .d\ 6\ G^{6.2} \tag{36}$$

$$C^1 \rightarrow cC^2. \quad d\ 5\ G^{6.2} \tag{37}$$

$$K \rightarrow A^1B^1C^1.D^1 \quad \dashv\ 0\ G^{6.2} \tag{38}$$

$$D^1 \rightarrow .dD^2 \quad \dashv\ 7\ G^{6.2} \tag{39}$$

Passo $i = 8$, conjunto de estados S_8 e $X_9 = d$

$$D^1 \rightarrow d.D^2 \quad \dashv\ 7\ G^{6.2} \tag{40}$$

$$D^2 \rightarrow .dD^3 \quad \dashv\ 8\ G^{6.2} \tag{41}$$

Passo $i = 9$, conjunto de estados S_9 e $X_{10} = d$

$$D^2 \rightarrow d.D^3 \quad \dashv\ 8\ G^{6.2} \tag{42}$$

$$D^3 \rightarrow .d \quad \dashv\ 9\ G^{6.2} \tag{43}$$

Passo $i = 10$*, conjunto de estados* S_{10} *e* $X_{11} = \dashv$

$$D^3 \rightarrow d. \quad \dashv \ 9 \ G^{6.2} \tag{44}$$

$$D^2 \rightarrow dD^3. \quad \dashv \ 8 \ G^{6.2} \tag{45}$$

$$D^1 \rightarrow dD^2. \quad \dashv \ 7 \ G^{6.2} \tag{46}$$

$$K \rightarrow A^1B^1C^1D^1. \quad \dashv \ 0 \ G^{6.2} \tag{47}$$

$$S \rightarrow K\{AdP(K)\}. \quad \dashv \ 0 \ G^{6.2} \tag{48}$$

$$\phi \rightarrow S. \dashv \quad \dashv \ 0 \ G^{6.2} \tag{49}$$

Ao aplicar a operação *scanner* no último estado não processado, estado 49, é adicionado ao conjunto de estados S_{11} o seguinte estado:

$$\phi \rightarrow S \dashv . \quad \dashv \ 0 \ G^{6.2} \tag{50}$$

Ao chegar ao fim do processamento de um conjunto de estados resultando em um estado S_{i+1} contendo apenas o estado 50, o algoritmo se encerra indicando reconhecimento da cadeia de entrada. A árvore de derivação pode ser montada percorrendo os estados em que foi aplicada a operação *completer*.

V. Considerações Finais

A implementação do algoritmo de Earley adaptativo apresentado no presente trabalho possibilitará a análise sintática de gramáticas adaptativas, sendo o primeiro componente que fará parte do novo arcabouço GrammarLab adaptativo.

O novo algoritmo traz como vantagens ser de propósito geral, podendo atuar em diferentes gramáticas sem que seja necessária uma implementação para cada uma, o que seria necessário caso fosse utilizado um autômato. A memória necessária para a execução da versão adaptativa do algoritmo de Earley será maior do que a utilizada em sua versão atual, uma vez que serão mantidas em memória diferentes gramáticas, que podem crescer exponencialmente de acordo com o tamanho da gramática inicial, de sua ambiguidade e da quantidade de ações adaptativas presentes na constituição. Além disso, na implementação atual do algoritmo no arcabouço GrammarLab, como forma de otimização de tempo de execução, a gramática é pré-processada, sendo gerados códigos fontes estáticos que devem ser compilados. Uma vez que a inclusão de adaptatividade requer um comportamento dinâmico, a estrutura atual do GrammarLab deve ser alterada, possivelmente representando um desafio para a implementação da abordagem proposta.

Como trabalhos futuros, o arcabouço continuará sendo evoluído de forma incremental, incluindo funcionalidades como a geração de classificadores baseados em gramáticas adaptativas, estimação de probabilidades de gramáticas adaptativas estocásticas e classificação de sequências e estruturas de RNAs contendo pseudonós.

Referências

[1] P. B. Menezes, Linguagens Formais e Autômatos: Volume 3 da Série Livros Didáticos Informática UFRGS. Bookman Editora, 2009.

[2] M. V. M. Ramos, J. J. Neto, and Í. S. Vega, Linguagens formais: teoria, modelagem e implementação. Bookman Editora, 2009.

[3] N. Chomsky, "On certain formal properties of grammars," Information and control, vol. 2, no. 2, pp. 137–167, 1959.

[4] A. K. Jain, R. P. W. Duin, and J. Mao, "Statistical pattern recognition: A review," IEEE Transactions on pattern analysis and machine intelligence, vol. 22, no. 1, pp. 4–37, 2000.

[5] J. Earley, "An efficient context-free parsing algorithm," Communications of the ACM, vol. 13, no. 2, pp. 94–102, 1970.

[6] D. B. Searls, "The linguistics of DNA," American Scientist, vol. 80, no. 6, pp. 579–591, 1992.

[7] M. Brown and C. Wilson, "RNA pseudoknot modeling using intersections of stochastic context free grammars with applications to database search." in Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing, 1995, pp. 109–125.

[8] D. B. Searls, "Linguistic approaches to biological sequences," Computer applications in the biosciences: CABIOS, vol. 13, no. 4, pp. 333–344, 1997.

[9] E. Rivas and S. R. Eddy, "The language of RNA: a formal grammar that includes pseudoknots," Bioinformatics, vol. 16, no. 4, pp. 334–340, 2000.

[10] M. Sipser, Introduction to the Theory of Computation. Thomson Course Technology Boston, 2006, vol. 2.

[11] D. H. Younger, "Recognition and parsing of context-free languages in time n3," Information and control, vol. 10, no. 2, pp. 189–208, 1967.

[12] J. J. Neto, "Adaptive rule-driven devices-general formulation and case study," in International conference on implementation and application of automata. Springer, 2001, pp. 234–250.

[13] M. K. Iwai, "Um formalismo gramatical adaptativo para linguagens dependentes de contexto," Departamento de Computação e Sistemas Digitais (PCS)-Escola Politécnica, Tese de Doutorado, Escola Politécnica, Universidade de São Paulo (USP), São Paulo, SP (in portuguese), 2000.

[14] J. J. Neto, "Adaptive automata for context-dependent languages," ACM Sigplan Notices, vol. 29, no. 9, pp. 115–124, 1994.

[15] A. Machado-Lima, "Laboratório de geraçao de classificadores de seqüências," Master's thesis, Universidade de São Paulo, 2002.

Gilmar Pereira dos Santos possui graduação em Sistemas de Informação pela Universidade de Mogi das Cruzes (2006), especialização em Engenharia de Software pela Faculdade Impacta de Tecnologia (2012). Atualmente é aluno de mestrado do Programa de Pós-Graduação em Sistemas de Informação da Universidade de São Paulo, atuando nos seguintes temas de pesquisa: linguagens formais, reconhecimento de padrões, classificação funcional de RNAs, análise de sequências, aplicação de métodos adaptativos em problemas de bioinformática.

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 S ão 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.

Ariane Machado Lima possui graduação em Ciência da Computação pela Universidade de São Paulo (1998), mestrado em Ciências da Computação pela Universidade de São Paulo (2002) e doutorado em Bioinformatica pela Universidade de São Paulo (2006), pós-doutorado pela Universidade de São Paulo. Atualmente é Professora do Curso de Sistemas de Informação da Universidade de São Paulo e orientadora nos seguintes programas de Pós-Graduação: Sistemas de Informação e Interunidades em Bioinformática. Tem experiência na área de Ciência da Computação, com ênfase em Ciência da Computação, atuando principalmente nos seguintes temas: linguagens formais, reconhecimento de padrões, bioinformática, RNAs não codificantes, análise de sequências.

Adaptivity: One Phenomenon, multiple perspectives of study

Rosalia Caya and João José Neto

***Abstract*—This article studies adaptivity as a general phenomenon, a subject of matter, permeable to different perspective within the research areas. From simple and general definitions grounded in Biology and Psychology to sophisticated and particular definitions within Computer technology areas, such as Computer Science, Information Technology, and Software Engineering, this work focus in retrieving the core features that characterize adaptivity, its needs, the different connected areas and the characteristic problems it is suited to solve. We present a proposal for the holistic characterization of adaptive behavior that contributes in the establishment of a framework to share scientific knowledge regarding specification, representation, modeling, design and implementation of adaptive behavior in the technological arena.**

***Index Terms*—Adaptivity, self-* systems, complexity, cybernetics, autonomic computing.**

I. Introduction

THE idea of an entity capable of modifying its own behavior according to characteristics of its environment and its own particular goals it's not a novelty, as a matter of fact it has been around from ancient times. From Daedalus, the most ingenious inventor of Greek myth, credited with making the first "living statues" that appeared to be endowed with life because of its capability to make human-like movements, wept and even vocalized [1]. Automata, from Greek perspective self-operating entities made to obey particular goals[2], were also engineered by Hephaestus, the Greek god of invention and technology. Talos, the gigantic animated bronze warrior programmed to guard the island of Crete, was one of Hephaestus's creations. Many variations of the myth exists and the analysis of the story of Talos has risen several books and studies. However, most of them agree on some characteristics of Talos: autonomous entity, with an specific task given by a superior authority that needs to be aware of the circumstances in its surroundings to be able to fulfill its goal, through the acquisition of hidden knowledge or underlying truth [3]. This seem to presage today's scientific "cybernetic organism"[1].

Automatic machines were also created by Italian inventor Leonardo da Vinci. Leonardo's robot (or Leonardo's mechanical knight) was a humanoid automaton designed and possibly constructed by Leonardo da Vinci around the year 1495. The robot knight is capable of performing several human-like motions: he could stand, sit, raise its visor and independently maneuver its arms. t is partially a result of Leonardo's anatomical research in the Canon of Proportions as described in the Vitruvian Man focusing in the complexity of the movements in human body.

A new attitude towards automata is to be found in Descartes when he suggested that the bodies of animals are nothing more than complex machines. France in the 17th century was the birthplace of those ingenious mechanical toys that were to become prototypes for the engines of the Industrial Revolution following a reductionist approach. This way of thinking culminated in unprecedented economic growth and development, and machines entered in everyone's lives. However, techniques continue to evolve along with technology and computing power, so, new models can be developed that better reflect the real world and its complexity [4]. Most real-world problems deal with complexity, uncertainty and optimization of some type where information must be exploited as acquired so that performance maintains or improves apace. This characteristics compound the basic definition for adaptive behavior we will present in the next section, and permeates problems in several areas of knowledge as diverse as ecology, psychology, economy, artificial intelligence, computational mathematics, sociology, and others. This way, over the years, some fields within the technological arena have dedicated special efforts to study adaptive behavior and developed approaches to deal with it in its particular domain.

However, in the last decades the study of adaptivity is gaining attention as a multi-disciplinary concern due to the rising demand for intelligent and more realistic applications. Applications mimicking human behavior, considering continually changing conditions, critical systems or high-definition simulation of real life situations have introduced back into technology the complexity cut off by the reductionist approach at the beginning of computer's era. This opens the door for many potential applications that require real-time perception and reaction. In fact the rising of new applications empowered by technologies, such as Internet of Things, Ubiquitous Computing, Multi-agent Systems, Evolving Systems, Cyber-Physical Systems, autonomic computing and others demand the support of some sort to process adaptive behavior while aiming a particular goal.

As consequence a variety of spaces to develop research related to adaptivity have been created. Some institutions and research groups specifically address adaptive behavior, complexity and dynamic change. Well established venues on technology have incorporated issues on adaptivity within their main tracks as academic work and research projects related to the topic continue to rise, as detailed in [5]. In fact, to deal with this growth dedicated venues from different natures:

R. Caya was with the Department of Electrical and Computer Engineering, Polytechnic School, University of São Paulo, São Paulo, SP - 05508-010, Brazil, e-mail: rosalia.caya@usp.br.

Manuscript received September 29, 2017.

academic (workshops, conferences, and journals) and mainstream (magazines, blogs and trade journals) have been created to cover topics on adaptivity. The same can be observed within the industry where inclusion of assisted technology and human-centered approaches welcome variety, complexity and uncertainty in every-day technology.

However, despite the growth and advances shown by research, this scenario and promising tendency reveal one major challenge: the need for a unified approach for adaptivity. The aforementioned approaches taken by specific fields present ad-hoc solutions to deal with adaptivity within their domains. On one side, this situation allows that different areas respond with autonomy to the challenges as they appear, creating new knowledge process and understandings. On the other side, the same situation is an ideal environment for creation of domain specific terminology, methods, models and resources, that sometimes remain invisible or even conflict between each other. We believe that by facing that challenge researchers collaborate to highlight the multi-disciplinary nature of adaptive behavior and empower the growth of a unified field.

To address this challenge, in this article we propose an study on literature of four different fields to create an holistic framework matching theoretical and practical approaches to define, model and create technology considering adaptive behavior.

Paper Organization The rest of the paper is organized as follows: first, in Section II, we analyze the historical scientific interest in the study of change processing within technological fields. In Section III we look at the approaches this phenomenon has taken within different technological areas aiming to solve problems related to their particular subject matter. We took four main fields within technological arena related to adaptivity that have developed solid approaches for dealing with adaptive behavior: self-* systems, cybernetics, complex systems, and autonomic computing. In Section IV, we propose an holistic vision of adaptivity, the factors that allow such approach, the common ground between the theory in the analyzed fields, and the contribution and advantages it will bring. Later, in Section V, we took a revision of related works on unified initiatives for adaptive behavior and highlight the difference they present with this work. Finally, in Section VI, we elaborate the conclusions about this work and present the future directions to develop an holistic vision of adaptivity.

II. Adaptivity: the phenomenon

The evolution and grown of technology have developed systems more complex and heterogeneous. At the same time, the interaction with such systems, made that the demand from users for mechanisms that allow personalization, reconfiguration, flexibility and autonomy passed from preference to necessity. When consulting literature about computer science and technology one can find several studies that highlight a novel ability from the systems to adjust themselves to events in their surrounding. Systems with this ability can be called *"self-adaptive systems"* within software engineering, or *"dynamically adaptive systems"* for the dynamic change community, or as part of *"autonomous"* or *"evolving"* systems. As matter of fact, there have been several keywords, over the years and across disciplines, that aim to describe some aspects of interest of a wider phenomenon: adaptive behavior.

A. The intuitive definition

In the most basic approach, we define adaptivity in the following intuitive terms:

Adaptivity is the ability of an *entity*, at any moment, to *decide* the modification, by *executing* a set of proper actions, of its own features, structure and/or behavior, or even its environment, when facing new coming events *perceived* in its surroundings while pursuing a particular *goal* to *suit more efficiently* the new *context* of its functioning.

The term *entity* can take a wide range of meanings for example: individual, system, structure, agent, being, and so on, encompassing different magnitudes from a single molecule to an interacting group of organisms. The concrete meaning is largely determined by the field of study. The *decision* to effectively change any of its components comes uniquely from its own analysis about the benefits gained in doing so. The direct *execution* of the proper set of actions matching the situation can have indirect repercussions in other available actions that define the behavior of the entity. The analysis mechanism is performed due to stimulus *perceived*, or sensed, in the current situation. The high-level *goal* the entity aims generally is set by an external high-level authority: a leader, a manager, a need or even evolution. The motivation in performing adaptations is a better suiting of the entity and its goal into the new situation, this have a variety consequences, such as: incorporation/drop of new features to take advantage of opportunities, avoidance of threats, or preparation for facing danger. By being adaptive the entity tries to respond to changes in the *context* in which it performs at the moment. The context is define by its internal features (behavior and structure) the characteristics of the external environment (resources, events, objects within) and the channels of interaction between them.

III. Adaptivity: the multiple perspectives

Adaptive behavior is an ability that has been studied within technological and computational arena for a long time from different perspectives. Most of this perspectives correspond with a particular domain, and were developed taking into account the focus of the field, the terminology and the concepts proper of that field. However, as we will see, when analyzing adaptivity they present profound similarities. We will present four main fields in technology that study adaptive behavior: Self-* systems, Cybernetics, Autonomic Computing and Complex Systems. For each of them we will give a concise definition of the field, it main focus, the general characterization of its architecture and it relationship with adaptivity.

A. Self-* Systems

A self-* system is a computer system that maintains at least one aspect of its operation automatically to relieve some of the

burden that complex systems put over human administrators. The aspects it automatize are known as self-* properties, they are a set of features that characterize the behavior of some complex system [6]. The *self-* prefix highlight the autonomous nature of this property, meaning that the system has the power to decide, perform and control over this feature on its own.

These properties, even when particularly applied to describe technological entities, were inspired and observed first in natural organisms [7] whom use them efficiently to achieve particular goals or overcome difficulties. Thereby, the self-* properties are related to adaptive behavior by describing the requirements and consequences of applying adaptivity within the systems. Figure 1 shows a hierarchical categorization of some of the most referenced self-* properties over the years. From these self-* properties two of them differentiate in nature from the others: self-management and self-adaptivity. Self-management is a vision, it can be thought as the parent or superclass of all self-properties. It describes a system that has at least one self-* property [8], [9]. Self-adaptivity or self-adaptiveness is the ability of an entity to evaluate its own behavior and change it when the evaluation indicates that its not accomplishing what the software is intended to do [10]. So self-adaptivity it can be understood as ability to process dynamic change in an autonomous and intelligent way, adjusting some characteristics in its behavior according to the environment in which the system performs. Therefore, we can say self-adaptivity is subsumed by other, more narrow self-* properties [8].

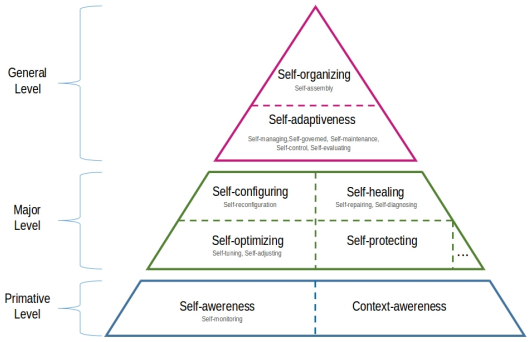


Fig. 1. Hierarchy of Self-* properties. Adapted from [11], [12], [9], [13]

Properties at the primitive level of this hierarchy, self-awareness and context awareness, are necessary condition for any system to process some type of change. Properties at the major level embedded more complex functions, hence, they are provided with more advanced mechanism of evaluation and control.

The four properties highlighted are the more referenced of the major level in the literature of different fields related to adaptive behavior. They are explicitly named as fundamental for the Autonomic Computing of IBM [14]. However, since the 2001 launch of Autonomic Computing by IBM, the set of self-* properties has grown substantially[15] and other self-* properties have been defined in the last years for computer systems[16]. In [6] and [9] the authors present a detailed, however not closed or definitive, list of self-* properties together with example of some types of systems implementing them.

Properties at the general level are those of higher complexity present in a system that processes changes and they are usually performed and observed by more sophisticated entities that control functionalities in different parts of the system. It is important to point out that in literature the term self-adaptivity has several terms closely related, particularly, self-managing and self-governing are used interchangeable. The attention, study and analysis in self-* properties, particularly self-adaptive, correspond to the interest of several areas of knowledge in dealing with problems with complexity and difficult to manage with traditional approaches [17], [18].

Characterization of Self- Systems:* As Self-* Systems permeates a wide set of properties, and their particular behaviors, the concrete architecture of a self-* system may vary from one to another depending of which ones are implemented. Moreover, a general approach for self-* systems can only point out the basic elements, conditions and mechanism, looking more as a guideline than an architectural model. Today self-* systems draws upon several well developed technologies that may be used as building blocks[19]. The idea of programs that reason about their own behavior and are able to manipulate their own semantic representation at runtime is not new, in fact it's very well founded and developed. Reflection and exception handling mechanisms are techniques that can be found in the majority of Programming Languages and support the basic traits of adaptivity. Reflection has been around long enough for efficient implementation methodologies to be developed. Reflection provides the tools for writing such programs but it doesn't provide any guidance in how it should be done. To help addressing this challenge there are many other contributing technologies , such as: model-based computing, theorem provers, models for reasoning about uncertainty, and agent based systems to name a few. [19].

From design perspective, as pointed out in [20] three metaphors have been useful to early researchers on self-adaptive software: coding an application as a dynamic planning system, or coding an application as a control system, or coding a self-aware system. In each case, self-* systems features are mapped into the paradigm's structure (planning system, control theory, or self-aware) aiming that some valuable insights and techniques can be borrowed to self-adaptive systems.

The work in [21], [22] sum up some of the effort performed by Software Engineering community to analyze models and patterns that can be applied to develop self-adaptive systems.

B. Cybernetics

Cybernetics is, in general terms, defined as the science that studies the abstract principles of organization in complex systems, focusing in how systems use information, models, and control actions to steer towards and maintain their goals, while overcoming difficulties [23]. By being inherently interdisciplinary, cybernetic reasoning can be applied to understand systems of any kind and it has influenced many fields included computer science, robotics, management, sociology, political science, economics, psychology and philosophy.

The field of cybernetics was created after WWII by a group of intellectuals interested in studying ”circular causal and feedback mechanisms in biological and social systems” to develop a general theory of organizational and control relations in a system [23].The first use of the term in English was at 1948 by mathematician Norbert Wiener in its seminar book *”Cybernetics: Control and communication in the animal and the machine”* [24]. There, Wiener states that Cybernetics is the study of **communication and control** in animals and machines, communication being the receiving and digesting of information, and control the use of this information in a direct action. Wiener's approach is in fact mechanistic, correspondent to the kind of machines used back on the days, and its known as Cybernetics of the 1st order or first-order Cybernetics.
According to Beer in [25]: ”cybernetics studies the flow of information round a system, and the way in which this information is used by the system as a means of controlling itself: it does this for animate and inanimate systems indifferently”. Later, organization theorists regard Cybernetics as a science of information processing, decision-making, learning, adaptation, and organization, whether this occurs in individuals, groups, organizations, nations, or machines [26].

After the Control Engineering and Computer Science disciplines become fully independent, some remaining cyberneticians felt the need to differentiate from the mechanistic approach by emphasizing autonomy, self-organization, cognition, and the role of the observer in constructing the model of a system. In this approach the system being study is interpreted as an agent in its own right, interacting with another agent, the observer. Hence, the observer too is a cybernetic system, trying to construct a model of another cybernetic system. To understand this process, we need a ”cybernetics of cybernetics”, i.e. a ”meta” or ”second-order” cybernetics. Therefore, 2nd. Order cybernetics, tries to understand adaptive autonomy and, further, shifting adaptability.

Nowadays, the second order perspective is firmly ingrained in the foundations of cybernetics overall. It is undeniable that many of the core ideas of cybernetics were assimilated by other disciplines, or inspired the development of new contemporary fields, and hence, continue influencing scientific developments. More generally, the philosophy of cybernetics is starting to permeate popular culture.

1) Cybernetics Model: According to [27] the operation of cybernetic systems can be characterized by a cycle with five stages, as shown in Figure2:

1) goal activation, this stage is about discovering goals, intentions and expectations that can be achieved, in the current state of the system. The triggering of this stage is the perception of an event (any kind of stimuli, disturbance, perturbation) from the environment through the system's sensory mechanism. Selecting the features of a system to pay attention to is inherit in the perceptual process;
2) action selection, this stage enacts the decision making mechanism of the system. Based on the goal it pursues this stage select the suiting plan or strategical actions, depending of the repertoire available, for achieving it;
3) action, this stage is responsible for the execution of the strategy, the implementation of the necessary context and the detailed directions for the actual behavior of the system in the environment;
4) outcome interpretation, at this stage the system retrieves the data and facts from the environment to interpret the consequences of the behavior performed before. This information is passed to the system as feedback using memory updating and storing. Some of the criteria used to elaborate the feedback are: effects on the environment, collateral effects in the system, and nature of the outcomes (long term vs short term, abstract vs concrete, approach vs avoidance);
5) goal comparison, this stage is about examination, deliberation and revision of the information from the feedback to reveal the achieving of the selected goal. If the goal has not being achieved yet, then some actions may be done to approach its completion. This actions can be towards to take advantage of an opportunity, or for avoiding a threat. Finally, some actions promoting changes in the system may be requested. This can be sum up as analyzing the mutual influence between the system and the environment in which it performs.

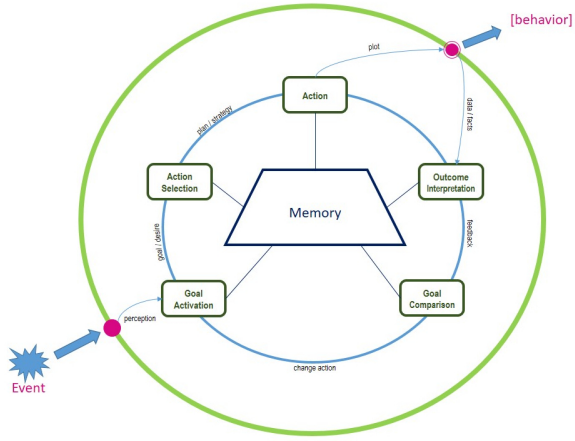


Fig. 2. Cycle in Cybernetic systems

These elements can be divided into two basic categories:

- First,there is a collection of mechanisms that evolved to carry out the different processes associated with each stage of the cycle. The mechanisms called into play by encounter with uncertainty are of two fundamental types, reflecting the unique status of the unknown as the only class of stimuli that is simultaneously innately threatening and innately promising: Stability and plasticity. The first of these needs is to maintain the stability of ongoing goal-directed functioning. The second is the need to engage in exploration that integrates novel or anomalous information with existing knowledge. Stability reflects variation in the control mechanisms that prevent the cybernetic system from being disrupted by emotional impulses.Stability reflects the capacity of the cybernetic

system to resist disruption. The term ”plasticity” is probably most often encountered in neurobiology, but here we understand plasticity as the general tendency toward exploration, with exploration defined as the creation of new goals, interpretations, and strategies [27]. All exploration involves learning).Plasticity reflects the degree to which the cybernetic system is prone to generating new goals, interpretations, and strategies, not only when required by stressors that have caused instability and disintegration, but also voluntarily, in response to the incentive reward value of the unknown;

- Second, stored in memory is a collection of goals, actions, and knowledge about the world (strategies, standards, behavioral repertoire, and patterns that exist in the world). Most of these are learned through experience rather than innately preprogrammed. These learned, updateable memory contents of the cybernetic system are deployed by the mechanisms described in the first category. Goals, interpretations, and strategies represent the information used by the cybernetic system to function in any situation, and they always reflect the manner in which the individual has adapted to that situation, even if they are one-off, never repeated. This means that not all adaptations are characteristic. To be considered ”characteristic,” the adaptation must have enough stability to be a useful descriptor of the person for some reasonable length of time.

2) Cybernetics and Adaptive behavior: Following the approach given by Wiener in [24] cybernetics can be also understood as the study of goal-directed self-adaptive systems [27], [28], [29]. By being a general theory of the regulation of systems it can be use as a framework to study management and organizations, encompassing adaption, self-organization and reflexivity. Cybernetics, as mentioned before, had a crucial influence on the birth and developing of various modern science [23]. Particularly, complexity, self-organization, self-reproduction, autonomy, networks, connectionism and adaptation were concepts first explored by cyberneticians during 1940's and 1950's [23].

All cybernetic systems receive feedback, through some kind of sensory mechanism, indicating the degree to which they are moving toward their goals and use this information to adapt and adjust their behavior, to pursue their goals. As explained in [26], Adaptivity is related to improvement, better suiting the conditions of an environment or enhance the performance of an entity. Ashby's theory of adaptation explains the success of process improvements methods based on the distinction between working IN a process and working ON a process. Work IN a process refers to the work done to make the process function. Work ON a process is the ”discussion” about how to improve the process. Ashby show that any system having two nested feedback loops, one inside the other, would be able to display adaptive behavior. The inner loop operates frequently and makes small adjustments, the outer feedback loop operates infrequently and initiates the learning of a new pattern of behavior. Adaptation encompasses learning.

C. Complex Systems

In both Engineering and Science the term *complexity*, does not have a sharp definition and the demarcation with the notion of complicated systems is a challenge [30], [31], [32]. However, this does not forestall a rigorous approach to the subject matter.

A system is considered complex if it have many components that collaborate to create a functioning whole. The function of such system is governed by the dynamical interactions of the components (within the system and with the environment) and cannot be fully understood by the description and analysis of its parts in an individual manner, as with the reductionist approach [33]. Complex systems are characterized, and distinguished from *complicated* systems, by two factors: (i) it exhibits unexpected behavior, often referred as emergence of properties, as consequence of non-linear interactions between its parts, framed by the hierarchical structure that build up the system, and (2) the uncertainty in predicting the behavior of the system, named unpredictability, due to continuous change in function and structure [30], [33]. The description of such a system can be done in different levels and from different perspectives, which means that **complexity** is subjective and it describes the stance that is being taken towards a system [31], hence the definition becomes unclear and arbitrary.

Traditionally, literature on complexity has tended to come from scientific domains taking a systems perspective, such as social and natural sciences, biology, sociology, philosophy which have frequently question emergent behavior, adaptivity and evolution. However, other areas of knowledge have turn to complexity as a way to get answer at questions that would otherwise remain inaccessible and to offer a key to new kinds of understanding [30]. Therefore, Complexity has become a highly interdisciplinary topic today, building bridges between several fields. This reflects the fact that most real-world systems are complex, and so increasingly are our technologies. As pointed out in [34] complexity is considered an inherent feature of the matter, being nature its ultimate source[35], and so is technology.

According to [30] the field of complexity studies has split into two subfields: the study of *Complex Physical systems*(CPS) and the study of *Complex Adaptive Systems*(CAS). The former studies has a set of tools and questions centering on elements with fixed properties and has let to a better understanding of physical phenomena. The latter deals with elements that are not fixed, usually called agents, that learn or adapt in response to interactions with other agents, continually exchanging information.

1) Characterization of Complex Adaptive Systems: Complex system, due to its study from different perspectives, present a challenge when trying to organize the features involving its functioning and structure. In [36] Holland proposes a general way of charactering Complex Adaptive Systems by seven basics: four properties(p): aggregation, nonlinearity, flows and diversity, and three mechanisms(m): tagging mechanism, internal models and high-level reorganization patterns. These seven principals intent to be a framework to organize the different features present in complex adaptive systems.

They are not the only basic features that could be selected from all the complex systems, although, most of the other can be derived from appropriate combinations of them. Actually, literature present different analysis of complex systems characteristics according to models or basic theory. Another approach to analyze is presented in [34] where the author elaborates table summing up the basic Properties of Complex Systems and its comparison with other systems.

In the following is a list, even though not exhaustive, we collected the main characteristics identified in Complex Systems from seminar literature in the field [30], [37], [36], [32], [33]:

- Self-Organization: as the ability to structure and restructure themselves, to learn, diversify and increase their complexity;
- Decentralization of decision making;
- Nonlinear relationships or interactions between its components: a small change in the system can lead to disproportionate effects;
- Learning capacity: a computational mechanism by which a cognitive system could iteratively build up a detailed and hierarchical model of its environment;
- Default hierarchical structure to organize and manage the behavioral laws at different levels. A hierarchical structure is the natural consequence of the aggregation property mentioned in [36];
- Feedback mechanism: reinforcing learning actions are self-enhancing and lead to better performances or avoid threads in future behavior;
- Path dependency and historical information: the state of a complex system at any point in time depends on the sequence of events and decisions that preceded that point. This way learning and action mechanisms would allow the system to adapt without losing what it had learned in the past;
- Emergent behavior;
- Balancing two forces or operations: one reinforcing growth by taking advantage and keeping already known successful configurations (exploitation), and the other aiming at discovering new combinations of traits that can retrieve better building blocks yet unknown (exploration);
- Mindset and models: all complex systems creates and use internal models to prosper: This models can be of different nature: tacit or explicit, learned in a single lifespan or through evolution. Models represent both: the own adaptive agents and the environment in which it perform. It is common the models interact with stored knowledge as a way to bring efficiency to the functioning of the system;
- Adaptive interactions: the agents of complex system can perform adaptive actions to process change through behavioral rules continually adjusted through evolution and learning;
- Perpetual novelty, it is unlikely for it to reach an optimal or equilibrium.

2) Adaptation within this perspective: Some of the aforementioned characteristics even when not mentioning adaptivity directly depend upon this faculty to develop a technique for processing continuous change. Feedback, learning, self-organization, aggregation and balancing the dichotomy of forces are some of the characteristics at systems' level that laid upon adaptivity to achieve its major goals.

The direct mention of adaptivity is particularly granted to elements of the system, the agents. Agents in CAS have three levels of action:

- Performance: designates an agents' behavioral repertoire at a point in a time. It is often modeled as a set of rules and signals, and described via signal-processing approach. An agent is sensible to its environment via detectors and effectors.

Adaptation happens by changing the signal-processing rules, which corresponds to changes in the structure of the associated network. To implement such a change the CAS needs that agents be able to:

- Credit assignment: an agent requires a means of assigning a quantity (strength, grade), that rates the usefulness of different rules in helping the agent to attain important resources;
- Rule discovery: a mechanism to discover new rules by combination and/or reorganization of atoms from successful rules. The new rule is biased by the agent's previous experience.

In [36], Holland states that adaptation is the *sine qua non* of CAS. He defines CAS as systems composed by interacting agents described in terms of rules. These agents adapt by changing their rules as experience accumulates. Moreover, in [37] the authors state that lifelong research work developed by Holland, recognized that adaption is central to fields that concern populations of agents that must continually obtain information from uncertain, changing environments and be able to use it to improve its performance and chance of survival, this extend the impact of studying adaptive behavior beyond CAS.

This approach of complex system and adaptivity grant the system with the freedom of working with several techniques to implement the required mechanisms.

D. Autonomic Computing

Autonomic Computing is a term coined by IBM's vice-president, Paul Horn at 2001, while presenting the idea in a keynote speech to the National Academy of Engineers at Harvard University[38]. It describes systems that can manage themselves, self-managing systems, given high-level goals from human administrators[39], [40], [14]. The term resembles the ability observed in the *Autonomic Nervous System* to govern and regulates a whole set of body functions, such as heart rate, body temperature, blood circulation and other "involuntary" actions without the need for conscious human involvement [39], [38], [41].The IBM initiative was formalized in [39], [14], [40] as a Grand Challenge within the information society, and particularly in the field of Information and Communication Technology (ICT) due to the complexity produced as byproduct of evolution in human society via automation. In particular, computer systems and its applications, have

experience an explosive growth in the last decades and now nearly pervade every aspect of our life [41]. However, they have reached the point in which the benefits that IT aims to provide are threaten by its own complex infrastructure, making them difficult to manage and use [39].To cope with this challenge, autonomic computing systems need to be capable of running themselves, adjusting to varying circumstances and preparing their resources to handle the workloads that we put upon them [39].

To build such high-level systems the authors in [39], [14], [40] describe the 8 key characteristics shown in Fig. 3 and detailed as follows.

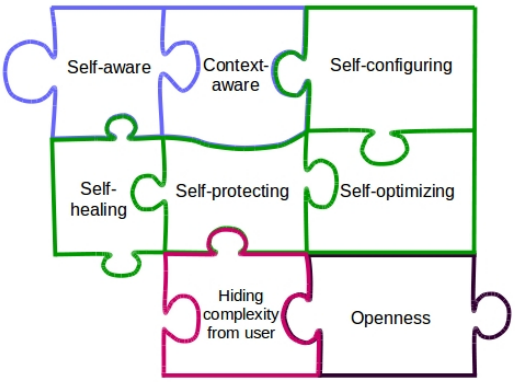


Fig. 3. Key characteristic of Autonomic Computing Systems. Adapted from [39], [14], [40].

In detail:

- Self-awareness and context awareness are inherent characteristics of Autonomic Computing systems regardless of the specific task each one performs. Any Autonomic Computer system must be able to observe their own structure an behavior to adapt or modify it, and be aware of its environmental and operational context [42].
- Self-configuring, self-healing, self-optimizing and self-protecting are the core capabilities that support self-management in Autonomic Computing. The works on [39], [40], [38] offer detailed descriptions and analysis of this capabilities.
- Hidden the complexity from user which implies autonomy in the process of decision taking to achieve the high-level policy given by the user.
- Openness, meaning that an autonomic computing system must be designed to operate in an heterogeneous environment, interacting with other technological elements (autonomic or not)[41]. Additionally, autonomic computing systems must be portable across multiple platforms [38].

The essence of Autonomic Computing initiative is to build self-managed, the intent of which is to free system administrators from low-level technical task, such as configuration, installation, updating and other system operation and maintenance tasks [40], to focus in high-level activities. It's worth notice that the automatic systems only performs the tasks that IT professionals choose to delegate to technology through policies.

1) Autonomic Computing Model: In [14] IBM describes a proposal for the necessary architectural building blocks, organization and behavior to build self-managing autonomic capability. The proposed architecture has two levels: systems' level and autonomic manager's level. At system's level the architecture organizes the elements in a hierarchy, as shown in Figure 4, governed by the manual managers, or IT specialists and assisted by knowledge sources.

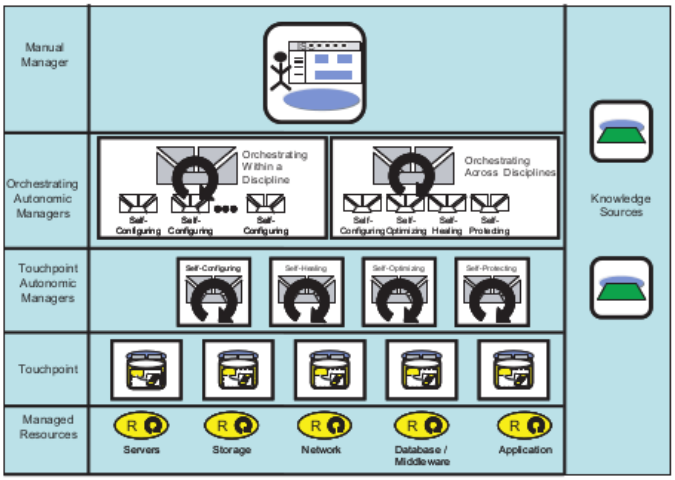


Fig. 4. Autonomic Computing reference architecture. Extracted from [14].

Layer 1, contains the managed resources (hardware/software, local/distributed), which may have embedded self-managing attributes. The resources, its types and scopes, define a set of decision-making context that are used to classify the purpose and the role of a control loop within the autonomic computing architecture. Layer 2, add a standard interface for accessing and controlled the resource called *touchpoint*. Layer 3 contains the touchpoint autonomic manager that interact directly with the touchpoints of managed resources to embody different tasks that support self-managing capability. Layer 4 contains autonomic managers that orchestrate other managers, they are the responsible for delivering the system-wide autonomic capability by implementing control loops that have the broadest view of the infrastructure.

At autonomic manager's level, the components must implement the architecture known as MAPE-K(Monitoring-Analyzing-Planing-Executing-Knowledge) architecture. Figure 5 shows the MAPEK architecture and its general elements. According to IBM, an autonomic manager must have the following features to exhibit self-managing properties [14]:

- An automated method to collect the details it needs from a managed resources, via touchpoint sensor interface and correlate them into symptoms that can be analyzed. This function is named *Monitoring*;
- An automated mechanism to observe and analyze situations to determine if some change needs to be made. To do so, in many cases, these mechanisms model complex behavior to employ prediction techniques. These mechanisms allow the autonomic system to learn about the

environment and predict future behaviors. This function is known as *Analyzing*;
- A mechanism that constructs the actions to enact a desired alteration in the managed resource, known as change plan, and logically passes that set of actions to the execute function. This mechanism can take many forms, goes from a single command to a complex work-flow. This function is known as *Planning*;
- A mechanism to schedule, control and carry out the actions on the plan change over one or more managed resources using the touchpoint effector interface of a managed resources. Finally, this function is known as *Executing*.

These mechanisms communicate and collaborate with one another and exchange appropriate knowledge and data. The information within the knowledge database consists of particular types of data (policies, symptoms, metrics, and logs) that can be access and/or modified by the mechanisms in the autonomic manager.

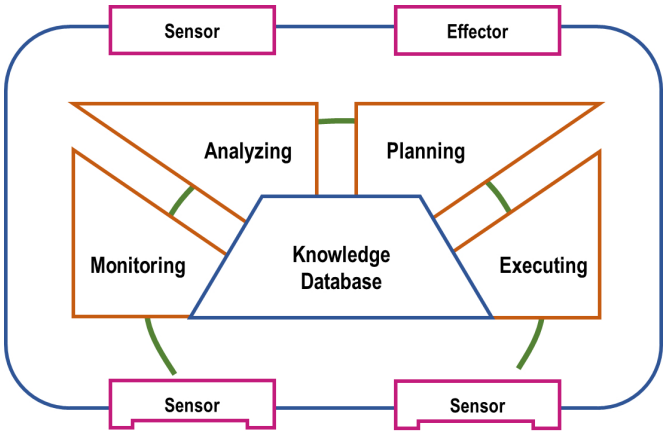


Fig. 5. Internal architecture for autonomic managers. Extracted from [14].

2) Adaptivity within Autonomic Systems: Adaptivity is the ability that supports the architecture of Autonomic Computing at both, system and component level. At a general level, the focus of Autonomic Computing approach is the design and implementation of self-management systems, which, according to [8], inherently laid upon adaptivity as main function. At the system's level the ability for rearranging components, within and between levels, exercises adaptivity to create, update or drop relations at run-time and as information arrives. Moreover, the self-* properties mentioned in the key characteristics for Autonomic Computing again laid onto adaptivity to process change. Finally, at component level, mechanism and elements organize too the basic elements to implement adaptive behavior. So adaptivity is present at Autonomic Computing as the backbone function to process dynamic change.

IV. Holistic vision

In general terms an adaptive program can be describe as a computer program that is able to evaluate its own behavior and make some changes in its own configuration (structure and functionality) at runtime when, due to new conditions perceived in its environment, its evaluation indicates :(i) that the program is not accomplishing its reason to exist, or (ii) that better functionality or performance is possible.

The variability of such a description allows formulating problems in different areas that involves optimization made difficult by substantial complexity and uncertainty where information must be exploited as acquired so that performance maintains or improves apace. Problems with this characteristics are pervasive and occur at critical points in different fields, as we have seen in the previous section, and even when the formulations to solve those problems can have a variety of guises, they give rise to the same fundamental questions.

A unified theoretical framework provides opportunities for identifying common interactions, methods and difficulties faced when studying adaptivity.

In the following subsections we will present the common ground of the four technological approaches to deal with adaptive behavior. We will draw the necessary conditions for the existence of adaptive behavior, the main properties observed in adaptive systems, and the characteristic of the problems that can found benefits in the use of adaptive behavior.

A. Necessary Conditions

- Awareness[43], [33], [44] : both self-awareness and context awareness. The former allows the program to inspect its own current configuration and recognize new needs and evaluation. The latter allows the program to perceive particular events in its application domain;
- Models[7] : system or internal model and environmental model. They can be either implicit and learned over evolutionary time, i.e. evolutionary computing, or explicit and learned (or given) over a single lifespan. The latter allows the program to reason about the impact of new-coming events in the environment in its own performance. Both models also allow the implementation of mechanisms of predictions that assist the program in decision taking during its functioning;
- Hierarchical structure[37]: hierarchy allows the application and managing of behavioral rules according to the level of abstraction of the entities and their capabilities;
- Characteristic response[37]: The normal or standard behavior that the program should perform under no changing conditions. This is the basic behavior over which the modifications, the adaptations, will be developed;
- Monitoring and selection Mechanism [44]: the mechanisms responsible for perceiving and filtering events in the environment and selecting plan and strategies to respond to them;
- Autonomy[43], [33]: the capacity to perform reasoning over its own behavior and control and acting to modify it without depending on human intervention;
- Learning Method [33], [44]: also called Learning Cycle. This is the part where the adaptivity *per se* is performed;
- Memory[37], [7]: stored knowledge.

Particularly, the learning method need to implement some basic elements:

- Feedback loop [44], [36]: the actions taken to modified the behavior and its consequences are incorporated in the stored knowledge, to generate better predictions and refine the desired behavior;
- Structural modifiers or operators [37], [7]: the elements responsible for implementing the direct modifications and its propagation. They implement the adaptive plan or strategy;
- Mechanism for comparison and assessment [44], [37], [7] : are the mechanism that allows the program to evaluate how convenient are different alternatives to modify its behavior.

B. Properties observed in Adaptive Systems

- Open-endedness [33], [37]: they are in dynamic stability, never achieve equilibrium due to balancing two opposite forces (exploitation vs exploration, or balancing loop vs reinforcement loop, or homeostasis vs homeorhesis).
- Continuity [33], [36], [7], [37], [44]: they are always perceiving changes;
- Traceability [30], [7];
- Self-* properties: usually adaptivity is used to manage the changes to achieve one or more self-* properties;
- Emergence and Propagation of features[44], [37];
- Exhibition of high-order patterns[37]: the mechanism and patterns used to implement adaptivity are not simple strategies and use sophisticated tools;
- Operation at multiple spacial and temporal scales.

C. Problem Characteristics

By identifying the main characteristics of a problem it is possible for a researcher to focus on searching resources (methods, tools, mechanisms and so on) that are aligned with the purpose of dealing with such characteristics. In this context, adaptive behavior has been credit as a good alternative in helping answering problems with the following main characteristics:

- Complexity [43], [32], [37], [7];
- Uncertainty [43], [32], [37], [7];
- Nonlinear interactions [32], [44];
- Changing Environments [37];
- An implicit optimization principle: the goal is to do the best by improving performance, and/or enhance/maintain the chance for survival with available resources [44], [37], [7].

V. Related Work

Related work naturally draws from several fields due to the multidisciplinary nature of adaptive behavior. We address related work from an unified vision of adaptivity. The most efforts to create an holistic approach comes from Complex Systems field and lately from Self-Adaptive Software Engineering. In complex System, Holland efforts for building a general theory of adaptation in complex systems are reflected in [36], [30], [37]. Moreover, the Santa Fe Institute[45] is one of the top institutions to research complex systems with a broader approach. The authors in [31] present another effort in develop a unified theoretical and practical framework for Complex Systems. Other researchers have taken the formal languages approach as a base to generate an unified model for adaptive software, such as [46], [47], [48], and other have tried to extend the concepts,models and architectures within their fields to cover different approaches, such as [41]. In [49] with can observe a unified an hybrid approach between cybernetics and formal methods.

From the field of Software Engineering, some efforts have been made for the better understanding of adaptivity. In literature we can find some of the authors from different backgrounds presenting a vision of integration with fields such as control theory [50], multi-agent systems [51], active software and so on [19]. In [21], [21] the authors explore architectures and models of Software Engineering and Testing for adaptive systems. With the same purpose some researchers have develop design patters for adaptivity [52].

However, the lack for a general examination that helps researchers in mapping elements from one approach to the other is the gap we are trying to fill with this work.

VI. Conclusion and Future Directions

By the end of this work we have come to the following conclusions:

- In the same sense that was previously mention in [32] the building of a common ground framework, where elements intent to be in its basic form, allows the appropriation of concepts and results from other complex systems for the purposes of explaining this concepts within adaptive technologies.
- Understanding the participants involved in adaptive behavior allows to identify core components, even when named under different terminologies, that perform the same task. By doing so it is possible to establish parallels between theories and techniques, resulting in opportunities to share knowledge [35].
- This holistic approach based on aggregation gives freedom to the researcher to integrate techniques, mechanisms, algorithms and other technological resources to different frameworks, representation or architectures according to the particular kind of adaptive behavior it needs to perform.
- Cross-comparisons provide the advantage of putting in evidence some characteristics that in one field are subtle and hard to extract while in other are salient and easy to examine [36].

Future work in the effort for building an holistic understanding of adaptivity are:

- Standard terminology should be build up because the current domain specific terminology does not help the holistic approach by being ambiguous or sometimes overlapping.
- Exploration and incorporation of new approaches for adaptivity in other fields to empower and support the holistic approach. In this sense, this work pretend to be only a small but significant contribution to such common

approach from the perspective of computing, we enthusiastically encourage researchers from other areas dealing with adaptive behavior to take an holistic approach and complement this effort.

- Elaboration of scientific analysis from areas such as Bibliometrics, Semantic Analysis and Semiotics can bring highly valuable information about how adaptivity is understood and the meaning it carries, helping to refine the holistic model. This studies are beyond the scope of the present work, but we encourage researchers in these areas to contribute with such an specialized knowledge to a common approach.
- Study of models and approaches for different parts of software development besides design and implementation, such as: testing, benchmarks,and others.
- Experimentation with hybrid approaches by exercising component model approach and aggregation properties.

REFERENCES

[1] A. Mayor. (2012, March) The world's first robot: Talos. The World's First Robot: Talos, Wonders and Marvels. [Online]. Available: http://www.wondersandmarvels.com/2012/03/the-worlds-first-robot-talos.html

[2] K. Anthis. (1990, Apr.) Talos. Automatons: Robots of Legend. [Online]. Available: https://sites.google.com/site/automatonrobotsoflegend/mischeif/talos

[3] W. Hyman, *The Automaton in English Renaissance Literature*, ser. Literary and scientific cultures of early modernity. Ashgate, 2011. [Online]. Available: https://books.google.com.br/books?id=TAIkn8WtjnIC

[4] M. Mitchell, *Complexity: A Guided Tour*. Oxford University Press, 2011.

[5] D. Enke, "Part i: Adaptive systems," *Procedia Computer Science*, vol. 20, pp. 11 – 13, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050913010314

[6] S. Poslad, *Autonomous Systems and Artificial Life*. John Wiley and Sons, Ltd, 2009, pp. 317–341. [Online]. Available: http://dx.doi.org/10.1002/9780470779446.ch10

[7] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992.

[8] E. U. Warriach, T. Ozcelebi, and J. J. Lukkien, "Self-* properties in smart environments: Requirements and performance metrics," in *In Workshop Proceedings of the 10th International Conference on Intelligent Environments*. IOS Press, 2014, pp. 198–205.

[9] A. Berns and S. Ghosh, "Dissecting self-* properties," in *2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, Sept 2009, pp. 10–19.

[10] R. Laddaga, "Self adaptive software baa 98-12 due 1/27/98," DARAP Broad Agency Announcement, Discussion paper, Tech. Rep., 1998.

[11] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Trans. Auton. Adapt. Syst.*, vol. 4, no. 2, pp. 14:1–14:42, May 2009. [Online]. Available: http://doi.acm.org/10.1145/1516533.1516538

[12] J. S. Bradbury, "Organizing definitions and formalisms for dynamic software architectures," *Techinical Report*, vol. 477, 2004.

[13] M. Rahman, R. Ranjan, R. Buyya, and B. Benatallah, "A taxonomy and survey on autonomic management of applications in grid computing environments," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 16, pp. 1990–2019, 2011. [Online]. Available: http://dx.doi.org/10.1002/cpe.1734

[14] "An architectural blueprint for autonomic computing," IBM, Tech. Rep., Jun. 2005.

[15] R. Sterritt, M. Parashar, H. Tianfield, and R. Unland, "A concise introduction to autonomic computing," *Adv. Eng. Inform.*, vol. 19, no. 3, pp. 181–187, Jul. 2005. [Online]. Available: http://dx.doi.org/10.1016/j.aei.2005.05.012

[16] D. B. Abeywickrama and E. Ovaska, "A survey of autonomic computing methods in digital service ecosystems," *Serv. Oriented Comput. Appl.*, vol. 11, no. 1, pp. 1–31, Mar. 2017. [Online]. Available: https://doi.org/10.1007/s11761-016-0203-8

[17] R. Frei, R. McWilliam, B. Derrick, A. Purvis, A. Tiwari, and G. Di Marzo Serugendo, "Self-healing and self-repairing technologies," *The International Journal of Advanced Manufacturing Technology*, vol. 69, no. 5, pp. 1033–1061, 2013. [Online]. Available: http://dx.doi.org/10.1007/s00170-013-5070-2

[18] F. D. Macías-Escrivá, R. Haber, R. del Toro, and V. Hernandez, "Self-adaptive systems: A survey of current approaches, research challenges and applications," *Expert Systems with Applications*, vol. 40, no. 18, pp. 7267 – 7279, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417413005125

[19] P. Robertson, R. Laddaga, and H. Shrobe, "Introduction: the first international workshop on self-adaptive software," in *Self-Adaptive Software*. Springer, 2000, pp. 1–10.

[20] M. M. Kokar, K. Baclawski, and Y. A. Eracar, "Control theory-based foundations of self-controlling software," *IEEE Intelligent Systems*, vol. 14, no. 3, pp. 37–45, May 1999. [Online]. Available: http://dx.doi.org/10.1109/5254.769883

[21] B. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, Eds., *Software Engineering for Self-Adaptive Systems*, ser. Lecture Notes in Computer Science. New York: Springer Berlin Heidelberg, 2009.

[22] R. de Lemos, H. Giese, H. Müller, and M. Shaw, Eds., *Software Engineering for Self-Adaptive Systems II. International Seminar, Dadstuhl, Germany, October 24-29, 2010. Revised and Invited Papers*, ser. Lecture Notes in Computer Science. New York: Springer Berlin Heidelberg, 2013.

[23] F. Heylighen and C. Joslyn, "Cybernetics and second order cybernetics," *Encyclopedia of physical science & technology*, vol. 4, pp. 155–170, 2001.

[24] N. Wiener, *Cybernetics: Control and communication in the animal and the machine*. Wiley New York, 1948.

[25] S. Beer, *Decision and Control: The Meaning of Operational Research and Management Cybernetics*. John Wiley, 1966.

[26] S. A. Umpleby, "Cybernetics," in *International Encyclopedia of Organization Studies. Volume 1*. Sage, 2007, pp. 350–353.

[27] C. G. DeYoung, "Cybernetic big five theory," *Journal of Research in Personality*, vol. 56, pp. 33 – 58, 2015, integrative Theories of Personality. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0092656614000713

[28] J. B. Peterson and J. L. Flanders, "Complexity management theory: Motivation for ideological rigidity and social conflict," *Cortex*, vol. 38, no. 3, pp. 429 – 458, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010945208706804

[29] C. S. Carver and M. F. Scheier, *On the self-regulation of behavior*. Cambridge University Press, 1998.

[30] J. Holland, *Complexity: A Very Short Introduction*, ser. Complexity: A Very Short Introduction. Oxford University Press, 2014. [Online]. Available: https://books.google.com.br/books?id=lmygAwAAQBAJ

[31] C.-C. Chen and N. Crilly, "From modularity to emergence: A primer in the design and science of complex systems," University of Cambridge,Department of Engineering, Cambridge, UK, Tech. Rep., 2016.

[32] M. Mitchell and M. Newman, "Complex systems theory and evolution," *Encyclopedia of Evolution*, pp. 1–5, 2002.

[33] P. Cilliers, *Complexity and Postmodernism: Understanding Complex Systems*. Taylor & Francis, 2002. [Online]. Available: https://books.google.com.br/books?id=OeCEAgAAQBAJ

[34] F. Grabowski and D. Strzalka, "Simple, complicated and complex systems: the brief introduction," in *2008 Conference on Human System Interactions*, May 2008, pp. 570–573.

[35] "How can we share solutions to complex systems problems across domains and application areas?" Mar 2017. [Online]. Available: https://phys.org/news/2017-03-solutions-complex-problems-domains-application.html

[36] J. H. Holland, *Hidden Order: How Adaptation Builds Complexity*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1995.

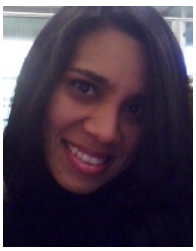
[37] S. Forrest and M. Mitchell, "Adaptive computation: The multidisciplinary legacy of john h. holland," *Commun. ACM*, vol. 59, no. 8, pp. 58–63, Jul. 2016. [Online]. Available: http://doi.acm.org/10.1145/2964342

[38] M. R. Nami and M. Sharifi, *A Survey of Autonomic Computing Systems*. Boston, MA: Springer US, 2007, pp. 101–110. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-44641-7_11

[39] P. Horn, "Autonomic computing: IBM's Perspective on the State of Information Technology," 2001.

[40] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003. [Online]. Available: http://dx.doi.org/10.1109/MC.2003.1160055

[41] P. K. Singh, A. Sharma, A. Kumar, and A. Saxena, "Autonomic computing: A revolutionary paradigm for implementing self-managing systems," in *2011 International Conference on Recent Trends in Information Systems*, Dec 2011, pp. 7–12.
[42] T. M. King, A. E. Ramirez, R. Cruz, and P. J. Clarke, "An integrated self-testing framework for autonomic computing systems," *JCP*, vol. 2, pp. 37–49, 2007.
[43] N. Khakpour, S. Jalili, C. Talcott, M. Sirjani, and M. Mousavi, "Formal modeling of evolving self-adaptive systems," *Science of Computer Programming*, vol. 78, no. 1, pp. 3 – 26, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167642311001742
[44] J. C. Principe, N. R. Euliano, and W. C. Lefebvre, *Neural and Adaptive Systems: Fundamentals Through Simulations with CD-ROM*, 1st ed. Berlin, Germany: John Wiley & Sons, Inc., 1999.
[45] S. F. Institute. (2017) Santa fe institute website. [Online]. Available: https://www.santafe.edu/
[46] R. Bruni, A. Corradini, F. Gadducci, A. Lluch Lafuente, and A. Vandin, *A Conceptual Framework for Adaptation*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 240–254. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-28872-2_17
[47] E. Serral, J. D. Smedt, M. Snoeck, and J. Vanthienen, "Context-adaptive petri nets: Supporting adaptation for the execution context," *Expert Systems with Applications*, vol. 42, no. 23, pp. 9307 – 9317, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417415005400
[48] A. Bhattacharyya, A. Mokhov, and K. Pierce, "An empirical comparison of formalisms for modelling and analysis of dynamic reconfiguration of dependable systems," *Formal Aspects of Computing*, vol. 29, no. 2, pp. 251–307, March 2017. [Online]. Available: http://dx.doi.org/10.1007/s00165-016-0405-z
[49] L. Chen, L. Huang, C. Li, and X. Wu, "Self-adaptive architecture evolution with model checking: A software cybernetics approach," *Journal of Systems and Software*, vol. 124, pp. 228 – 246, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121216000820
[50] Y. Diao, J. L. Hellerstein, S. Parekh, R. Griffith, G. Kaiser, and D. Phung, "Self-managing systems: a control theory foundation," in *12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'05)*, April 2005, pp. 441–448.
[51] C. Bailey, D. W. Chadwick, and R. de Lemos, "Self-adaptive federated authorization infrastructures," *Journal of Computer and System Sciences*, vol. 80, no. 5, pp. 935 – 952, 2014, special Issue on Dependable and Secure Computing. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022000014000154
[52] Y. Abuseta and K. Swesi, "Design patterns for self adaptive systems engineering," *CoRR*, vol. abs/1508.01330, 2015. [Online]. Available: http://arxiv.org/abs/1508.01330



Rosalia Edith Caya Carhuanina Was born in Lima, Peru. She received the degree of BsC (Bachelor in Science and Engineering) with major in Computer Engineer, and the professional degree of Computer and Systems Engineer from the Pontifical Catholic University of Peru (PUCP, Pontificia Universidad Católica del Perú) at Lima in 2009. Between the years 2007 and 2011, she participated of the activities at the Artificial Intelligence Research Group at the Computer Engineering Department of the same university. She holds a MsC degree in Electric Engineering, with major in the field of Computer Engineering from the University of São Paulo (USP) at São Paulo, Brazil. She is currently a PhD student developing her research in the Laboratory of Adaptive Languages and Techniques (LTA). Her main areas of research are: Adaptive Technology, Autonomous Systems, Cybernetics, Artificial Intelligence, Natural Language Processing, Human-Machine Interaction.

João José Neto graduated in Electrical Engineering (1971), Master in Electrical Engineering (1975), PhD in Electrical Engineering (1980) and Professor (1993) at the Polytechnic School of the University of São Paulo. Currently, he is an associate professor at the Polytechnic School of the University of São Paulo and coordinates the LTA - Laboratory of Languages and Adaptive Technology of PCS - Department of Computer Engineering and Digital Systems of EPUSP. He has experience in the area of Computer Science, with emphasis on the Fundamentals of Computer Engineering. He works mainly on the following topics: adaptive devices, adaptive technology, adaptive automata, and in their applications to Computer Engineering, particularly in adaptive decision making systems, analysis and processing of natural languages, construction of compilers, robotics, computer-aided teaching, modeling of intelligent systems, automatic learning processes and inferences based on adaptive technology

Reclassificação de Sentimentos em Tempo de Execução: Uma Implementação Adaptativa

Ariana Moura da Silva, Rodrigo da Matta Bastos e Ricardo Luis de Azevedo da Rocha
Escola Politécnica da Universidade de São Paulo - São Paulo - Brasil
Departamento de Engenharia da Computação - Laboratório de Tecnologias Adaptativas

***Resumo*—Através do processo de extração e automatização de *tweets* utilizando filtragem por *Emoji*, identificados como campo chave da tabela de unicode do Twitter [1], constatou-se que cada *tweet* pode expressar uma relação entre o *Emoji*/símbolo contido no texto, e o sentimento contido no *tweet* propriamente dito. Com a classificação do sentimento de cada tweet, o próximo passo é partir para a classificação do sentimento do assunto, ou seja, do *TopTrend* em questão. A proposta deste trabalho é exemplificar através da metodologia utilizando filtragem por *Emoji*, de um único tweet, que é possível realizar a classificação geral de um *TopTrend* utilizando a classificação de um conjunto de *tweets* deste mesmo *TopTrend*, e sugerir um modelo adaptativo capaz de reclassificar em tempo de execução o sentimento neste *TopTrend* conforme os indivíduos comentam sobre o assunto.**

***Index Terms*— Adaptatividade, PLN - Processamento de Linguagem Natural, Análise de Sentimentos, Emoji, TopTrend.**

I. Introdução

A ocorrência de acontecimentos oriundos de desastres naturais, ou até mesmo do lançamento de uma nova marca e/ou produto, mudanças de leis em nosso código penal, mudanças comportamentais de indivíduos, formalização de uma ideia ou opinião de um determinado assunto, transformam-se em notícias, comentários ou relatos que invadem nossos meios de comunicação [2]. Juntamente com esses acontecimentos e a externalização desses relatos documentados de forma textual utilizando *blogs*, redes sociais, *websites* entre outros.

Constatou-se que o individuo produz uma massa de dados muito grande, o que já chamamos da "Era Big Data"[3]. Devido a isso surge então as novas áreas de pesquisa voltadas para análise dessas informações documentadas, para posterior geração de conhecimento. Indo mais além surge a necessidade de conhecer o sentimento envolvido/expresso pelo individuo ao registrar o texto/menção. Afinal o ser humano é um individuo emocional, que exprime emoções inclusive na sua maneira de escrever.

O grande desafio encontra-se no propósito de como quantificar, metrificar e criar indicadores ou padrões de classificação de sentimentos capazes de entender a complexidade da língua e a forma de expressão do individuo [4]. Frente a este problema, o presente trabalho tem o objetivo de apresentar a relação entre o *Emoji* inserido no texto e o significado do texto. Técnicas de Processamento de Linguagem Natural e filtragem por Emoji são métodos utilizados para a classificação de sentimentos e avaliação da reação que a população exprime sobre o assunto analisado.

Nas demais subseções desta Seção I serão abordados os grandes temas: Adaptatividade, PLN, Análise de Sentimento e *Emoji*; a Seção II apresenta os objetivos do trabalho; a Seção III define a metodologia utilizada; e a Seção IV apresenta algumas conclusões obtidas a partir de análises em exemplos apresentados e possíveis trabalhos futuros.

A. Adaptatividade

Segundo a conceituação de automato adaptativo extraída da tese de doutorado de José Neto (1993)[5]:

> "A cada execução de uma transição adaptativa, ou seja, a cada ocasião na qual a máquina de estados que implementa o autômato adaptativo sofre alguma mudança em sua configuração, tudo se passa como se surgisse uma nova máquina de estados, caracterizando, para o autômato, a execução de um passo adicional em uma trajetória de reconhecimento do texto de entrada, em um espaço de máquinas de estados".

e juntamente com estudos realizados no trabalho Workshop de Tecnologias Adaptativas de 2016, algoritmos adaptativos vêm sendo empregados em processamento de linguaguem natural e para este trabalho em específico na reclassificação de sentimento em tempo de execução, justamente pelos recursos e característica que a adaptatividade permite.

B. PLN

A linguagem humana não é simplesmente a manifestação de uma ação física qualquer do ser humano. As palavras são como símbolos, em que seus significados indicam para uma ideia ou coisa. Os símbolos da linguagem podem ser codificados em voz, gesto, escrita e outros. Processamento de Linguagem Natural (PLN) tem diferentes níveis desde o processamento da fala até a interpretação semântica e processamento de discurso. PLN tem por objetivo projetar, construir algoritmos capazes de ajudar a máquina na compreensão na linguagem natural humana [6].

Tarefas como verificação ortográfica, pesquisa por palavra-chave, encontro de sinônimos, são tarefas consideras fáceis. Já a análise de conteúdo/informações em sites ou documentos são consideradas nível médio. As tarefas de nível difícil são: tradução automática, análise semântica, co-referência ou encontrar respostas subliminares em perguntas destacadas em determinado documento [6].

C. Análise de Sentimentos

Uma emoção não é simplesmente um estado de sentimento. Emoção é uma cadeia de eventos frouxamente ligados que começam com um estímulo e incluem sentimentos, alterações psicológicas, impulso para a ação e comportamento específico, dirigido por objetivos [7].

Análise de sentimento é uma técnica usada para extrair e encontrar automaticamente o sentimento expresso em linguagem natural. O termo sentimento refere-se a sentimentos ou emoções como nossos sentidos, audição, visão, toque, olfato e paladar. O que queremos extrair das mensagens compartilhadas nas redes sociais é o sentimento expresso de forma positiva, negativa ou neutra.

A análise de sentimento no nível da palavra verifica a polaridade desta palavra especificamente. No nível da sentença, será levado em conta não apenas o polaridade das palavras que a contém, mas também as relações entre essas palavras e seu uso gramatical. No nível do documento, leva em consideração o contexto completo do documento, levando a uma análise mais complexa sobre como as frases interagem umas com as outras [8].

D. Emoji

Atualmente os *Emojis* são muitos populares na escrita em redes sociais, principalmente no Facebook e também em aplicativos de troca de mensagens instantâneas, como o WhatsApp. É uma expressão de origem japonesa composta pela junção dos elementos **e** (imagem) e **moji** (letra), e é considerado um pictograma ou ideograma, ou seja, uma imagem que transmite a ideia de uma palavra ou frase completa.

A figura 1 mostra um exemplo de *Emoji* chorando de rir, apesar da palavra "chorando"possuir uma polaridade negativa quando observada individualmente, no sentido da palavra. Quando composta pela palavra "rir", a palavra "chorando"passa a ter um valor de exagero e juntas formam uma polaridade positiva. Levando em consideração que "Chorando de Rir"foi a etiqueta dada para o símbolo de *Emoji* demonstrado na figura.

"Nós usamos entonação de voz ou linguagem corporal para contextualizar o que estamos dizendo. Os *Emojis* são a forma que usamos para fazer isso online", afirmou Iyad Rahwan, professor associado do MIT [9].

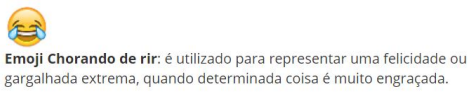


Figura 1. Exemplo de *Emoji* Chorando de Rir.

II. Objetivos

O objetivo principal deste trabalho é realizar a coleta automática de um determinado *TopTrend*, especificamente mensagens em língua portuguesa extraídas do micro blog Twitter. Armazenar as mensagens de forma sistemática, que permita a recuperação através de filtragem utilizando os *Emojis*. Realizar um exercício de classificação de sentimentos nos *tweets* de forma manual, observando a característica do texto e o sentido semântico daquela menção. Observar qual seria o percentual de cada faixa etária de sentimentos classificados. Descrever o modelo adaptativo que poderá ser empregado para que o próprio sistema possa se reorganizar e realizar a classificação do *TopTrend* em tempo de execução.

E o objetivo secundário deste trabalho é dar continuidade e apoio técnico ao trabalho apresentado no WTA 2016 (Análise Semântica de Sentimentos Utilizando Árvores de Decisão Adaptativa [10]).

III. Metodologia

Foram desenvolvidos programas na linguagem Java para a extração e armazenamento automático das mensagens em língua portuguesa extraídas do Twitter. Esse programas possuem a função de coletar mensagens sobre os *toptrends* (tópicos mais populares do blog) e também a partir de parâmetros de busca específico, informado pelo usuário. Podendo ser por data, uso de palavra-chave, ou popularidade da mensagem na rede. As mensagens extraídas são arquivadas em um banco de dados estruturado no programa *MySQL*, em conjunto com informações que permitam a sua filtragem posterior. Além das tarefas propostas, foi utilizado um método de Análise de Sentimento baseado na análise de *Emojis* [1], para ser aplicado nas mensagens extraídas pelos programas. A base de *Emojis* utilizada para análise foi a do *Unicode* disponível na url: http://www.unicode.org/emoji/charts/full-emoji-list.html.

Filtrou-se a partir do banco de dados apenas *tweets* que possuem *Emoji* em sua descrição, e que fazem cruzamento através de campo chave com a tabela universal do *Unicode*, já com a classificação simplística de polaridade, onde o *tweet* faça parte do *toptrend* 'GrandPrix'. Essa primeira filtragem foi proposta a partir da hipótese em que o *Emoji* contido no *tweet* tem relação direta com o texto propriamente dito.

A segunda filtragem, foi realizada a partir desta primeira amostra de dados importadas no Excel, reduzindo então o conjunto de dados de 4.083 para 335 registros. Utilizando o campo hora do *tweet* filtrou-se o intervalo das 10h00 até 11h00 do dia 06 de agosto de 2017, e também no campo *hashtag* deve conter também a expressão 'GrandPrix'.

O termo 'GrandPrix' refere-se ao torneio de vôlei feminino realizado em Nanjing (China), e neste dia 06 de agosto de 2017 a seleção brasileira de vôlei feminino jogava a final contra a seleção italiana de vôlei feminino. A seleção brasileira venceu por 3 sets a 2 (26/24, 17/25, 25/22, 22/25 e 15/8) e conquistou o título pela 12ª vez, enquanto que as italianas ainda não conquistaram nenhuma taça. Para a seleção italiana o jogo era um grande desafio, enquanto a seleção brasileira foi confiante para o jogo, porém no decorrer do jogo a situação mudou diversas vezes, sendo visível que o Brasil custou a vencer esta rodada [11].

O entendimento do contexto do evento que ocorreu no mundo real é importante, para que se inteirar da situação e apoio na interpretação da menção explicíta no tweet. Com certeza esse conhecimento dos fatos ocorrido durante a partida de vôlei é importante na classificação de sentimentos.

Dado o conhecimento do evento, realizou-se uma classificação manual dos 335 tweets. Os sentimentos utilizados

na classificação foram extraídos da Roda das Emoções de Plutchik [7]. A primeira classificação por polaridade (positivo, negativo e neutro) e segunda classificação de sentimentos divididos em:

- Positivo
  Admiração / Alegria / Otimismo / Sarcasmo / Serenidade / Surpresa
- Negativo
  Aborrecimento / Apreensão / Desaprovação / Medo / Nojo / Pensativo / Raiva / Sarcasmo / Tristeza
- Neutro
  Neutro

IV. Resultados

Através da tabela Unicode foi possível criar uma chave entre os códigos que simbolizam os *Emojis* do Twitter e a tabela de significado por polaridade. Com essa primeira filtragem foi possível quantificar as mensagens do *TopTrend* 'GrandPrix'.

A figura 2 exemplifica as polaridades obtidas de menções extraídas do Twitter no dia 06 de agosto de 2017 através da *hashtag* GrandPrix, que indica o evento Grand Prix de Volêi 2017 (http://grandprix.cbv.com.br/).

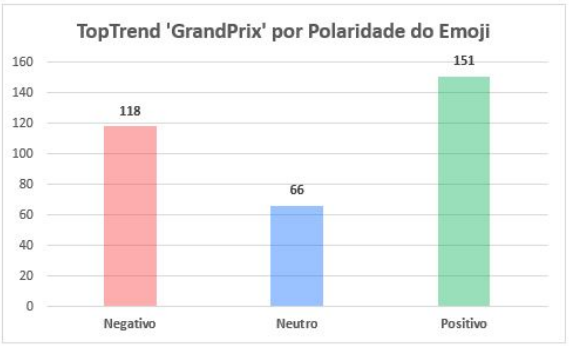


Figura 2. Polaridade do *Emoji* do *TopTrend* 'GrandPrix' dia 06/08/2017 das 10h às 11h

A figura 3 exemplifica as polaridades dos *Emojis* versus as polaridades obtidas na classificação manual. Dentro da polaridade Negativo pela classificação por *Emojis*, têm-se as polaridades Negativo, Positivo e Neutro. Onde Negativo possui um número de ocorrência de 87 tweets, indicando então que este é a maior porção, dentro da mesma polaridade Negativo, validando então que o *Emoji* tem relação direta com a menção do tweet. Em seguida a polaridade Positivo com 26 ocorrências, e a polaridade Neutro com 5 ocorrências.

Para a polaridade Positivo pela classificação por Emojis, o número de ocorrências da polaridade Positivo foi de 86, validando novamente que o *Emoji* tem relação direta com a menção do tweet, na sequência a polaridade Negativo com 57 ocorrências e a polaridade Neutro com 8 ocorrências.

Para a polaridade Neutro pela classificação por Emojis, o número de ocorrências da polaridade Neutro foi de 9, o que não segue a mesma metodologia de relação do texto com o *Emoji*. Na sequência dentro desta mesma classificação têm-se a polaridade Positivo com 17 ocorrências e a polaridade Negativo com 40 ocorrências.

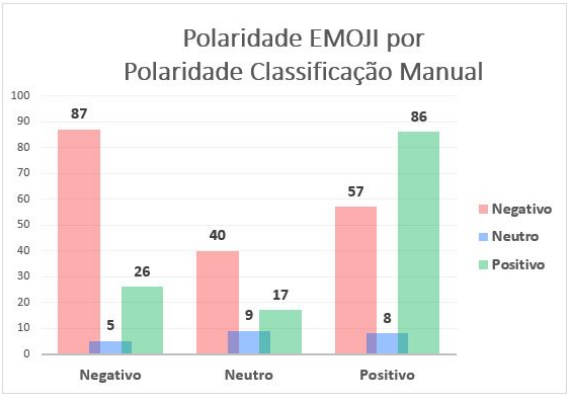


Figura 3. Polaridade Classificação por *Emoji* contido no **Tweet** versus Polaridade Classificação Manual do *Tweet* com referência ao texto escrito pelo usuário

Na polaridade Neutro, que não segue a mesma metodologia, olhando no detalhe a classificação dada pela Tabela Universal do Unicode, pode-se perceber a necessidade de uma atualização desta classificação. Na figura 4, propõem-se uma reclassificação de algumas categorias, baseado na classificação de sentimentos manual, e pelo significado que este *Emoji* representa. Neste caso, se fosse aplicada a reclassificação, as polaridades Negativo e Positivo seriam extintas na classificação por *Emoji* Neutro.

Polaridade Emoji	Significado Emoji	Emoji	Sugestão de Reclassificação do Emoji
Neutro	cara cansada	😫	Negativo
Neutro	cara de sono	😪	Negativo
Neutro	face neutra	😐	
Neutro	rosto aliviado	😌	Positivo
Neutro	rosto babando	🤤	Negativo
Neutro	rosto com os olhos arreganados	😬	
Neutro	rosto inexpressivo	😑	
Neutro	rosto pensativo	🤔	Negativo
Neutro	rosto perseverante	😣	Positivo
Neutro	rosto sem boca	😶	
Neutro	rosto sorridente	☺	Positivo
Neutro	rosto triste mas aliviado	😥	Negativo

Figura 4. *Emoji* classificados pela tabela universal Unicode que poderiam ser reclassificados conforme frequência de *tweets* com outras polaridades e sentimentos

A figura 5 mostra a classificação manual realizada no conjunto de 335 registros. Duas classificações foram realizadas, primeiramente a de polaridade e em sequencia a classificação dos sentimentos. Os sentimentos utilizados foram extraídos da Roda das Emoções de Plucthik.

A figura 6 relaciona os sentimentos mais significativos que representam 98% da base classificada de forma manual. Neste intervalo das 10h às 11h pode-se concluir que mesmo o Brasil ter ganho a partida, a maior parte dos *tweets* classificados foram de aborrecimento.

A partir da observação da figura 6, não pode-se concluir para o *TopTrend* 'GrandPrix' do dia 06/08/2017, que o conjunto de usuários brasileiros, mais os seus *tweets* escritos em língua portuguesa, representam uma conclusão de Aborrecimento. Na figura 7, pode-se observar que o para o sentimento Aborreci-

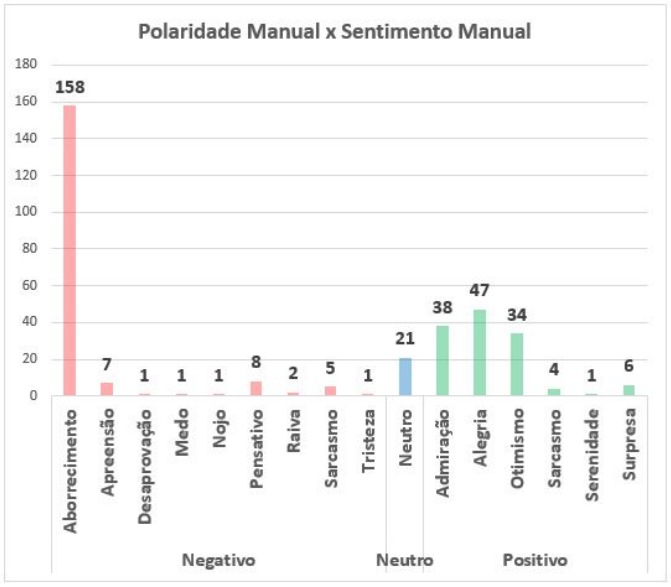


Figura 5. Classificação de Sentimentos Manual do *TopTrend* 'GrandPrix' dia 06/08/2017 das 10h às 11h

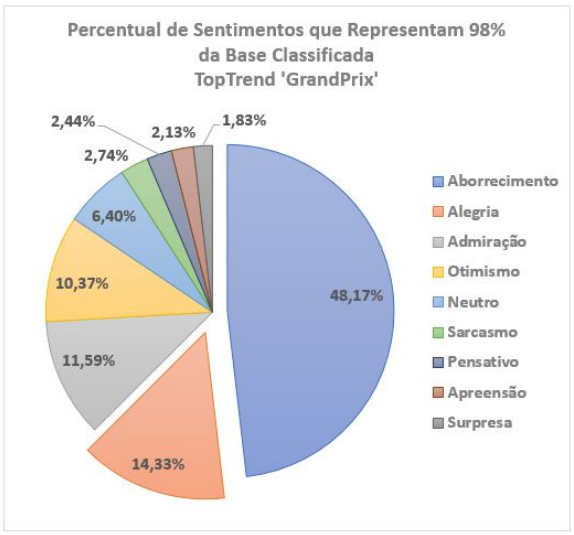


Figura 6. Sentimentos mais significativos, que representam 98% da base

mento existem alguns picos, que geralmente são nos erros de jogadas ou ponto da seleção italiana.

Na figura 8 realizou-se uma filtragem para remover os *retweets* do sentimento Aborrecimento. Pois uma outra hipótese observada é de que os usuários possuem uma tendência a republicar a opinião de um outro usuário com maior influencia na rede social (os chamados nós fortes[2]). A partir da retirada de *tweets* que não foram escritos pelo próprio usuário, ou seja, que ele apenas republicou o conteúdo de outro usuário, é possível observar a diminuição na frequência dos picos e o aumento de intervalo entre eles. Mostrando que realmente, os usuários possuem essa tendência em republicar conteúdos que possuem opinião forte na rede.

Na figura 9 outra filtragem foi aplicada. Considerou-se os últimos cinco minutos do conjunto de dados, que refere-se

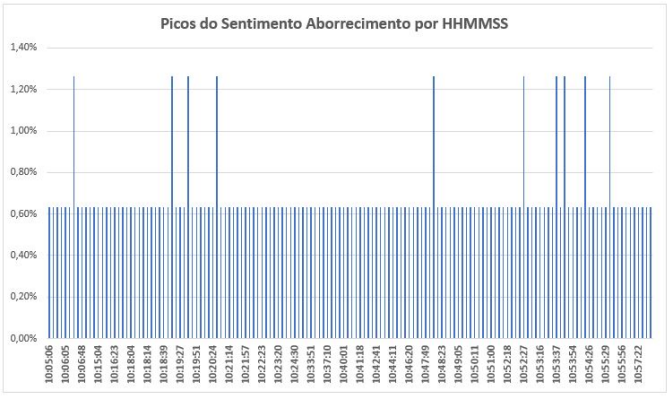


Figura 7. Picos do sentimento Aborrecimento por hora minuto e segundo

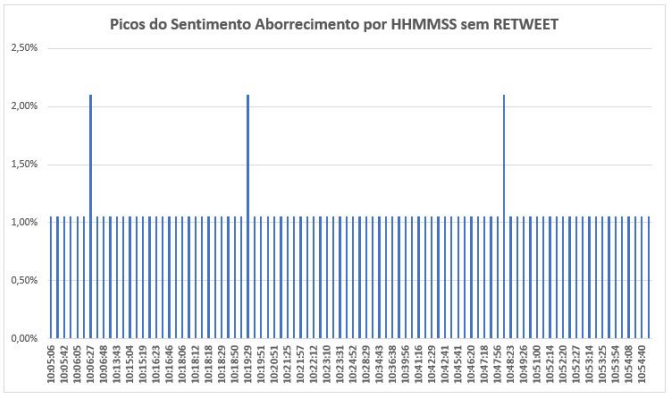


Figura 8. Picos do sentimento Aborrecimento por hora minuto e segundo sem retweets

ao contexto da virada de jogo, onde o Brasil começa a ter vantagem de pontos sobre a seleção italiana, no último sets. A partir desta análise, pode-se observar que mesmo boa parte do jogo os usuários relatando o nervosismo, medo, sufoco, desapontamento com o andamento do jogo, não quer dizer absolutamente que a classificação geral para este *TopTrend* seja fundamento na maior frequência do sentimento Aborrecimento, e sim no resultado final da partida.

Observando os últimos cinco minutos, o resultado é totalmente diferente, tendo um percentual de positividade muito maior, sendo representado por 23,81% de Alegria, 14,29% de Admiração e 28,57% de Otimismo. Enquanto que a taxa de negatividade sofre uma queda brusca, sendo representado por 14,29% de Aborrecimento e 9,52% de Apreensão.

Aplicando um modelo de classificação de sentimentos com um recurso Adaptativo implantado, que possa interpretar a política de contexto de um evento em tempo de execução, é possível mudar a classificação geral de um *TopTrend* que até então estava classificado de uma maneira, mas que devido a um evento inusitado mudou completamente o resultado da classificação geral do *TopTrend*. Como no caso do jogo de vôlei do Brasil contra a Itália. Eventos dessa natureza acontecem no mundo real, as situações de mercado financeiro, um evento na magnitude de desastre natural, as opiniões sobre produtos, podem ter seus resultados mudados a partir de um ou vários eventos conforme a sazonalidade e interferência na rede

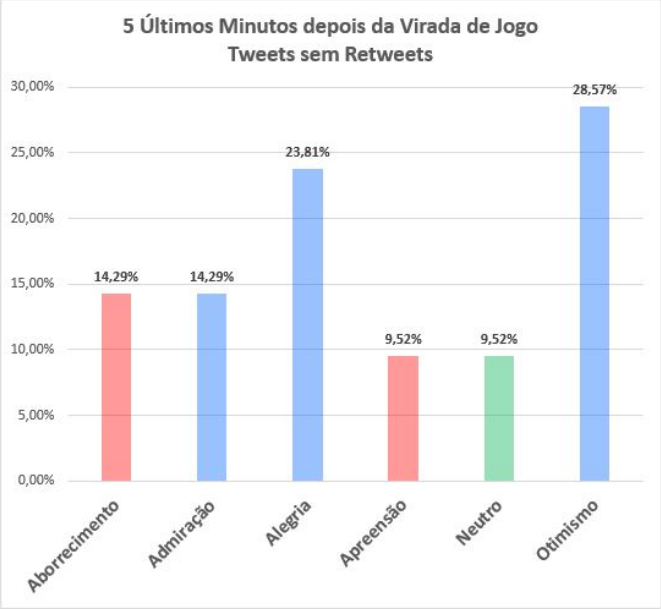


Figura 9. Cinco últimos minutos depois da virada de jogo, seleção brasileira em vantagem sobre a seleção italiana

de comunicação. Essas mudanças podem ocorrer alterando completamente a polaridade do *TopTrend*, e ocorrendo o que chamou-se de Reclassificação de Sentimentos em Tempo de Execução.

V. Trabalhos Futuros

Como sugestão de trabalhos futuros explorar mais a hipótese de *retweets* influenciarem no resultado do conjunto de dados, mesmo que o evento do mundo real represente outra classificação de determinado assunto. Classificar entidades semânticas para que se possa entender se o aborrecimento com o jogo refere-se ao resultado em si, ou a atitude de uma ou outra jogadora. Pois na leitura de alguns *tweets* muitos usuários opinavam sobre determinadas atitudes ou características físicas das jogadoras. Não deve ser descartada a possibilidade de realizar uma reclassificação também da tabela de *Emoji* universal do Unicode.

VI. Conclusão

Os programas conseguem realizar de forma automatizada etapas importantes para a Análise de Sentimentos em redes sociais. O experimento utilizando o método de análise por meio de *emojis* proporcionou vislumbrar um caminho para outros métodos. Em projetos futuros pretende-se desenvolver métodos de Análise de Sentimentos mais elaborados. É imprescindível a implantação de um módulo adaptativo para que possa ajudar na reclassificação de sentimentos em tempo de execução.

Agradecimentos

Os autores agradecem ao CNPQ processo no 141077/2015-8 pelo apoio recebido, na forma de concessão de bolsa de doutorado, para o desenvolvimento deste trabalho, à comissão organizadora deste Workshop, à coordenação do curso de pós-graduação em Engenharia da Computação e a Escola Politécnica da Universidade de São Paulo.

Referências

[1] SILVA, A. M. d.; BASTOS, R. M.; ROCHA, R. L. d. A. d. Análise de sentimentos de mensagens de redes sociais: Mineração de dados do twitter. In: *Anais do VI Workshop de Pós-Graduação da Área de Concentração Engenharia de Computação do Programa de Pós-Graduação em Engenharia Elétrica da EPUSP WPG-EC 2017*. [S.l.]: Departamento de Engenharia de Computação e Sistemas Digitais, 2017.

[2] SILVA, A. M. d. *REDES DE COMUNICAÇÃO DE DESASTRES NATURAIS: Indicadores Léxico-semânticos de Relevância Social em um Corpus Jornalístico*. Dissertação (Mestrado) — Universidade Federal do ABC, 2013.

[3] NAIK, K.; JOSHI, A. Role of big data in various sectors. In: *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*. [S.l.: s.n.], 2017.

[4] SILVA, A. M. d.; ROCHA, R. L. d. A. d. Análise de sentimentos de mensagens de redes sociais: Mineração de dados do twitter. In: *Anais do IV Workshop de Pós-Graduação da Área de Concentração Engenharia de Computação do Programa de Pós-Graduação em Engenharia Elétrica da EPUSP WPG-EC 2015*. [S.l.]: Departamento de Engenharia de Computação e Sistemas Digitais, 2015.

[5] NETO, J. J. *CONTRIBUIÇÕES À METODOLOGIA DE CONSTRUÇÃO DE COMPILADORES*. Tese (Doutorado) — Universidadede de São Paulo, 1993.

[6] CHAUBARD MICHAEL FANG, G. G. R. M. R. S. F. Natural language processing with deep learning. In: . [s.n.], 2017. Disponível em: <https://web.stanford.edu/class/cs224n/lecture$_notes/cs224n - 2017 - notes1.pdf$>.

[7] PLUTCHIK, R. The nature of emotions. v. 89, n. 4, p. 344–350, 2001.

[8] LIU, B. *Sentiment Analysis and Opinion Mining*. [S.l.]: Morgan & Claypool Publishers, 2012. ISBN 1608458849, 9781608458844.

[9] BRASIL, B. Emojis ajudam computador a identificar mensagens sarcásticas. In: . [s.n.], 2017. Disponível em: <http://f5.folha.uol.com.br/voceviu/2017/08/emojis-ajudam-computador-a-identificar-mensagens-sarcasticas.shtml>.

[10] SILVA, A. M. d.; NETO, J. J.; ROCHA, R. L. d. A. d. Análise semântica de sentimentos utilizando Árvores de decisão adaptativa. In: *Anais do X Workshop de Tecnologia Adaptativa - Área de Concentração Engenharia de Computação do Programa de Pós-Graduação em Engenharia Elétrica da EPUSP WPG-EC 2017*. [S.l.]: Departamento de Engenharia de Computação e Sistemas Digitais, 2016.

[11] BRASIL supera sensação italiana e conquista o 12º título de Grand Prix. https://esporte.uol.com.br/volei/ultimas-noticias/2017/08/06/brasil-x-italia-pela-final-do-granprix.htm: [s.n.], 2017.

Bio-Inspired Neural Network Applied to Urban Traffic Control in a Real Scenario

Nelson Murcia García
Department of Computer Engineering
Universidade de São Paulo, USP
São Paulo, Brazil
nelson.murcia@usp.br

André R. Hirakawa, Guilherme B. Castro
Department of Computer Engineering
Universidade de São Paulo, USP
São Paulo, Brazil

***Abstract*— Global economic development has the disadvantage of increasing the population in large urban centers that causes an increase of vehicles traffic in cities and results in traffic jams. Several researches have been developed to provide solutions to the problem of urban traffic. This work aims to apply the Bio-Inspired Neural Network model to control urban traffic semaphores. The proposed model was obtained and developed in previous works in the research group. Therefore, this paper presents the evaluation of the model applied on a real scenario of the big cities through simulations considering characteristics such as: vehicle speed, streets and avenues lengths, traffic semaphore positions and different traffic demands. The chosen scenario is the Paulista Avenue in the central region of city of São Paulo, well known for its high traffic demand. The Bio-Inspired Neural Network algorithm was compared with the fixed-time control, which is currently used for the control of semaphore phases in the city of São Paulo. The behavior of control algorithms were compared for low, average and heavy traffic demands. The performance indicators used were the average travel time of the vehicles and the level of occupation of the roads. The results show that the BiNN model is better in all the simulations made, with the different situations of traffic demands. Tests results show performance up to 44,67% in terms of average travel time, if compared to the fixed time control.**

Keywords—Bio-Inspired Neural Networks; Traffic Lights Control; Semaphore Control; Urban Traffic Control; Travel Average Time; Roads Occupation Levels.

I. INTRODUCTION

The development and growth of large cities today is governed by economy and social development. This explains how the urban population has increased significantly in recent years, according to [1]. In 2007, urban population exceeded the rural population, suggesting that by the year 2050, the urban population will represent 70% of the world population. Hence, the researchers present solutions that seek to improve the solutions to the problem of overcrowding in large urban centers. For this reason the researches are conducted to solve fundamental questions, in subjects such as: energy, sustainable development, security, housing, health and transport [2]–[4].

Transportation is a key issue for the urban development of cities. Solutions to problems related to this area are still a challenge for researchers. When the number of people in a space of similar size increases, vehicle traffic increases as well, being a serious problem faced today in large cities [5], [6]. Increasing the infrastructure of roads and highways, public transportation, and intelligent control at street crossings are some of the actions to be taken to solve transport problems. Semaphores are devices commonly used to control intersections, therefore, an optimum adjust of the time of each phase of traffic can help considerably as it could prevent traffic jams on the roads [7]–[13].

Many studies have been proposed to control the time of each phase at a semaphore. The unpredictable nature of traffic demand makes the task of optimizing the control more difficult. As a consequence, the most varied types of algorithms and methods are found in the literature, although modern urban traffic control can now be divided into two groups: Theory of Optimal Control and Artificial Intelligence. [8].

An adaptive semaphore controller is proposed by Taranjeet Kaur et al. in [13]. The controller uses neural networks and genetic algorithms to adapt the semaphore schedule according to the congestion of each intersection. The neural network receives the signal times as the input and provides the length of the queue as the output. Another example of the use of neural networks is presented in [10]. Authors proposed an optimum adjustment in the signal traffic times, concluding that most of the time, performances of the two proposed algorithms have a similar behavior, but significantly higher than the fixed-time controller. Another observation of that work is that the algorithm is proposed and tested only for a single semaphore crossing.

Finally, in [14], a Biological-inspired Neural Network (BiNN) is proposed and developed, being able to continuously monitor the system status and make decisions. The proposal establishes a multi-agent system concept and allows the coordinate control of several crosses. The vehicle queue size at the intersections of the streets is considered as the system entry variable. Furthermore, it proposes a method for determining parameters according to the desired behavior and provides a method for stability analysis. The algorithm is validated using an urban mobility simulator and compared to a conventional iterative controller. Results show better controlling behavior even in low, moderate and heavy traffic situations. This model extends to a multi-agent model where each agent controls a

single intersection of the street and interacts with the neighbor agents to achieve coordinated control of the various crossings. The proposal prevents the saturation of the streets and coordinates the activities of the neighboring agents causing green lights at the semaphores.

Following sections presents the application of BiNN algorithm for semaphore control in a real scenario. Section-II presents some definitions of BiNN algorithm and explains the equations that describe the model. Description of the scenario and the simulation setup is shown in Section-III and Section-IV respectively. Section-V presents and analyzes the results obtained during the simulation. Finally, some concluding remarks are presented in Section-VI.

II. BiNN ALGORITHM

BiNN was proposed and evaluated in [14] considering hypothetic traffic scenarios also testing its behavior in controlling dynamic systems. Indeed, authors analyzed its stability and adaptability. The BiNN adopts biological characteristics, such as inhibitory synapses and mechanisms of adaptation of neural networks. Therefore, in this case, BiNN show better performance than artificial neural networks, and has the advantage of not requiring training period. This model, according to [8] and [15], has low computational cost and the mechanism of short-term plasticity and the oscillatory behavior facilitates the change of semaphore phases. Another advantage of the proposed method is the continuous monitoring of traffic status and decision making.

BiNN equations for urban traffic control system of this paper is proposed in [15], and presented in equations (1), (2) and (3):

$$A_i^{t+1} = \sum_{j \in N_i} w_{xy} Q_j^t \quad (1)$$

$$O_i^{t+1} = \frac{1}{1 + e^{-m(A_i^t - S_i^t)}} \quad (2)$$

$$s_i^{t+1} = \frac{vO_i^t + S_i^t}{v+1} \quad (3)$$

Equation (1) determines the activation function A of a neuron i at time $t+1$ based on the weighted sum of its N inputs Q. Equation (2) is a sigmoid function whose slope is determined by m and represents the activation function of neurons. It generates the output O of a neuron, based on its activation A and the displacement s of its activation function, which represents the mechanism of adaptation of the model (intrinsic plasticity). The factor m only represents the slope of the curve. Equation (3) determines the displacement s of the activation function of a neuron i based on its output, v is the adaptation coefficient, which is a small constant value that determines the rate of adaptation of the neurons.

According to [7], this model is split in two parts:

- Control of the phase change of semaphore at an intersection.
- Coordination of intersections that is responsible for green waves.

A. Control of an intersection

The structure shown in Fig. 1 is used to control each one of intersection. Each set of neurons (p, h, q) represents a semaphore phase. Considering n intersections, the whore structure will have n neurons sets. The neurons $p_{1,2...n}$ are the excitatory ones, $q_{1,2...n}$ are the sensory neurons, $h_{1,2...n}$ are the interneurons and $q_{a,b,c,d}$ are the sensory receptors, which measure the occupation of the relative pathways, representing the inputs of the system.

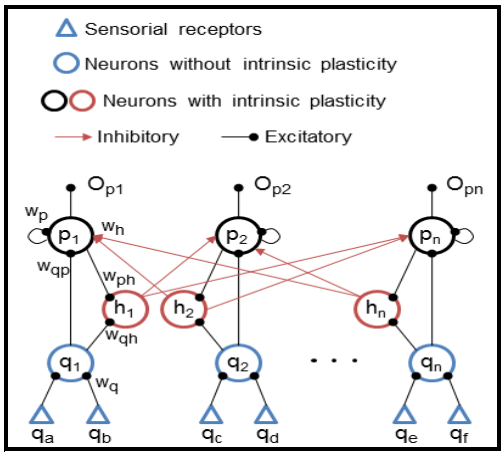


Fig 1. BiNN model structure for semaphore control

B. Coordination of intersections

For the coordination of intersections, the structure shown in Fig. 2 is used. The neuron of the semaphore phase 1 of intersection A is represented by $p_{1,A}$, while $p_{1,B}$ represents the same semaphore phase of intersection B. Hence, the traffic phases that control vehicle flows in the same direction are coordinated. Furthermore, $q_{a,A}$ and $q_{a,B}$ are the sensory receptors of intersection A and intersection B, respectively. All other neurons of the first structure are not considered in the control of intersections [14].

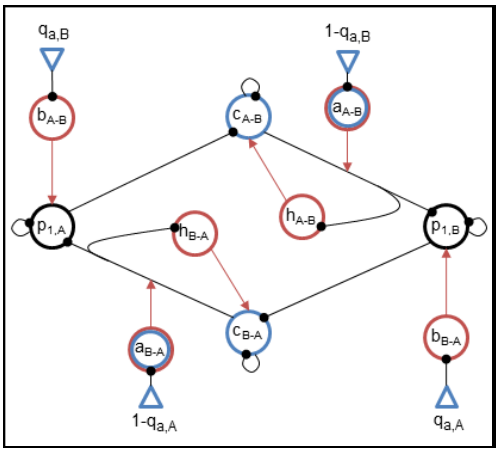


Fig 2. BiNN model structure for a semaphore coordination

The structure in the Fig. 2 has two basic operating principles:

- Storing the information when a semaphore has been activated until the corresponding semaphore of the next intersection is ready to become active.
- Inhibit the next semaphore-phase in cases occupancy of the next intersection is activated, as a way to avoid the overflow effect. This action is performed by the interneurons *b*, which inhibit the *p* neurons in the forward direction according to the occupation of the next intersection.

III. SCENARIO

The city of São Paulo had big problems with the level of traffic bottling. It is one of the largest urban settlements in the world, according to [16], with a population of 12,038,175 only in its metropolitan area, which has an extension of 1521.11 km^2, having a population density of 7,914.07 in habitants per square kilometers in its entire area. The city has a vehicular fleet of approximately 7 million vehicles and a road extension of approximately 17,000 km of streets and avenues [17]. As a result, São Paulo has one of the highest traffic levels, with frequently traffic jams of more than 150 km in the region of the expanded center at rush hour [17]. In such centre region, Avenida Paulista is one of the main avenues of the city, being the headquarters of the largest financial companies in the country, which means high concentration of people, as well as vehicles.

The selected scenario has a total of 4.4 km of streets and avenues, with an area of approximately 0.5 km^2, including 3 parallel streets and 4 cross streets (see Fig. 3), with 10 semaphores distributed in each intersection. This scenario has as border São Carlos do Pinhal street by the north and Alameda Santos street by the south, whereas the Alameda Joaquim Eugênio de Lima street and Teixeira da Silva street are the borders by the west and east, respectively. The distribution of semaphores is as follows: 4 semaphores in the Avenida Paulista, in its 4 intersections of this avenue with each transversal streets, and the other 6 semaphores in the streets that are to the north and south of this main route. There are 6 semaphores that include three phases, the four that are in Avenida Paulista plus two, besides the other four semaphores has two phases.

São Carlos do Pinhal street has only one-way traffic: east-west with two lanes in most of its extension, Avenida Paulista has two-way traffic, each one with three lanes: east-west and west-east, Alameda Santos street has two lanes in the west-east one-way traffic. Alameda Joaquim Eugênio de Lima has just a lane in a north-south way, Avenida Brigadeiro Luís Antônio has two lanes in both each north-south and south-north ways, Manoele da Nobrega street has two lanes in a south-north one-way traffic, Maria Figueiredo street has just a lane from north to south way and lastly, there is Teixeira da Silva street, which also has just a lane, but in a south-north way.

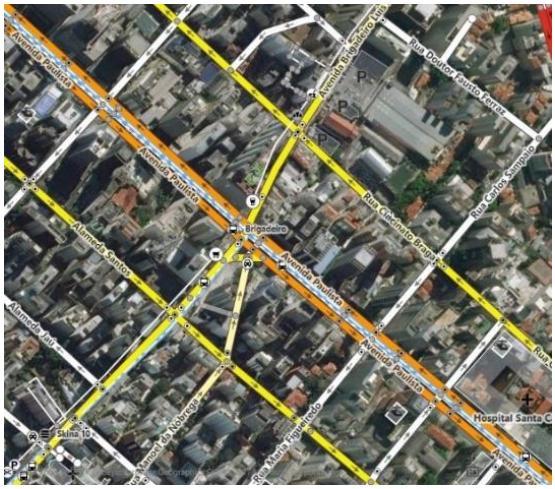


Fig 3. Map of chosen scenario

IV. SIMULATION

The simulations were performed using the MATLAB and SUMO ("Simulation of Urban MObility") tools (see Fig. 4) [18], [19]. The BiNN model was programmed in MATLAB, while the model of the urban transit system was programmed in XML ("eXtensible Markup Language") used in the SUMO environment. To perform the simulations and analysis of results, the protocol TraCI4Matlab [20] was used, which adopts the client-server paradigm and allows the interaction between SUMO (server) and MATLAB (client). All simulations have a duration time of 3600 seconds (one hour).

For analysis of the results, six different types of simulations (see table I), three simulations with different levels of vehicle demand for each type of control algorithm were performed, Fixed-time control and BiNN control.

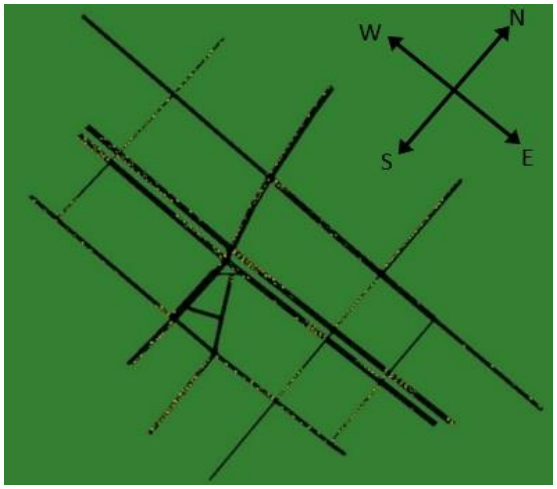


Fig 4. Simulation of scenario in SUMO

TABLE I. SIMULATIONS

Simulation Number	Simulations Descriptions		
	Control Algorithm	*Value (vehicles/ second)*	*Description*
1	Fixed Time Control	1.25	Low traffic
2	Fixed Time Control	2.50	Average traffic
3	Fixed Time Control	3.75	Heavy traffic
4	BiNN Control	1.25	Low traffic
5	BiNN Control	2.50	Average traffic
6	BiNN Control	3.75	Heavy traffic

The demand for 1.25 vehicles per second represents low traffic, while demand of 3.75 vehicles per second represents heavy traffic, so demand 2.50 represents a scenario with average traffic, as used by [8]. All vehicles used have the same characteristics, 5 meters long, with acceleration of 0.8 m/s^2 and maximum speed of 13.89 m/s, which is the speed limit in urban regions according to [17]. In addition, the stochastic steering behavior (SUMO simulator parameter) is equal to 0.5 in all of tests.

The routes of the vehicles were programmed with the same characteristics of the traffic routes described and observed in [17] and [21].

V. RESULTS

The performance indicators adopted are the mean travel time of vehicles, which was used in the analysis made by [14] and [22], and the level of occupation of the lanes in the scenario is the same as that was used by [8] and [9]. For calculating the average travel time of the vehicles was chosen to the east-west route of Avenida Paulista, which is the main avenue of this scenario and the city in general, and the average was obtained by measuring the travel time of 200 vehicles from one border to another of the select scenario. Fig. 5 shows the average travel time values obtained by the six types of simulations.

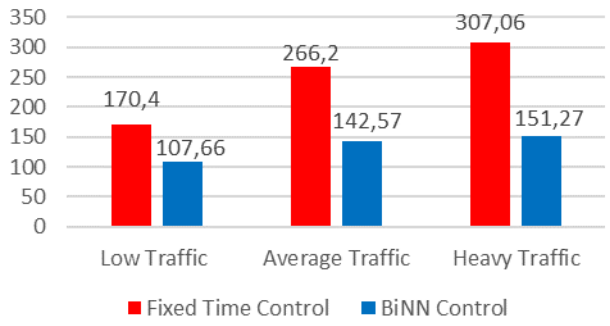


Fig 5. Average Travel Time of vehicles

For the low traffic level, the average travel time of vehicles with the BiNN control decreases by 62.74 seconds, which means more than one minute, which is a significant time for the analysis done on a stretch of avenue with 700 meters. For an average traffic level a difference of 123.63 was obtained, more than two seconds. Finally, for a heavy traffic level, the average travel time difference with the control algorithms applied was 155.79 seconds. We can conclude that the BiNN algorithm has a better response of traffic control than fixed times considering the average travel time. Moreover, its response improves with increasing traffic demand.

Regarding the level of occupancy of lanes, the simulations performed provided the results shown in Fig. 6, Fig. 7 and Fig. 8. The mean differences between the occupancy level of the lanes with the two control algorithms used are shown in table II. The reduction of this value can be justified with the results shown in the Fig. 9 and Fig. 10. While demand for traffic increases if the control designated to regulate traffic is not able to regulate the number of vehicles on the lanes, the scenario does not have enough space for new vehicles to begin their journey, therefore, the number of departed vehicles in the scenario with the average and heavy demand conditions and the fixed time control is lower than those that begin the route with the same demand and with the BiNN control. This situation is observed in Fig. 9, where the number of departed vehicles in the 6 simulations are shown. The difference of the numbers of departed vehicles in two types of control used in the simulation, in the case of the average demand level, is 1236 vehicles, while in the simulation with the heavy demand this difference is even greater, 1636 vehicles. Therefore, if the number of departed vehicles is greater and the control provides a better response to the system, the number of vehicles arriving has been increased using the BiNN control, as shown in Fig. 10.

With the analysis of the average travel time values explained above, we can infer that the BiNN control algorithm has a better response for the three types of simulated traffic demand.

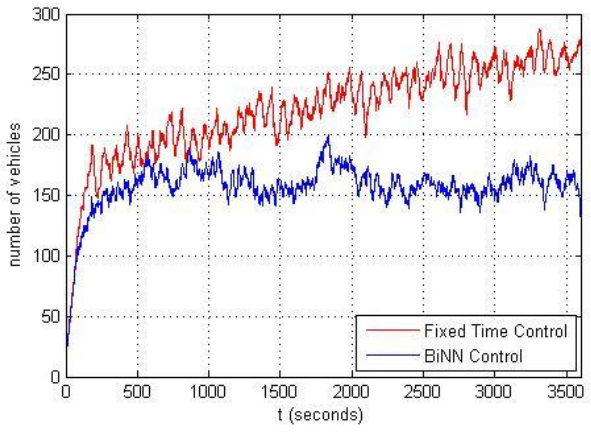


Fig 6. Simulation with low traffic demand

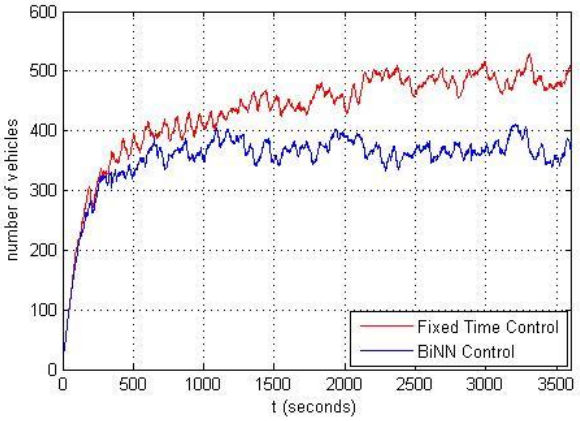


Fig 7. Simulation with average traffic demand

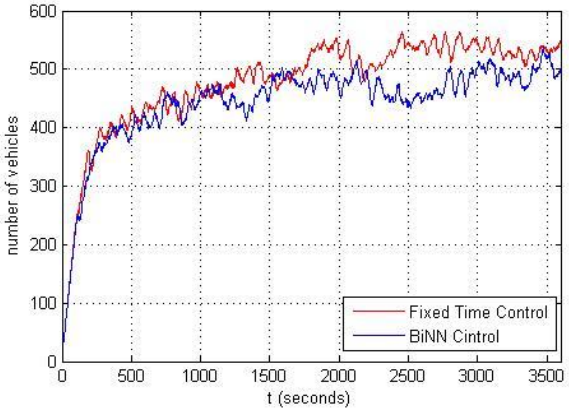


Fig 8. . Simulation with heavy traffic demand

TABLE II. DIFFERENCE AVERAGE IN ROAD OCCUPATION LEVEL

Traffic demand	**Difference Average** (number or vehicles)
Low traffic	62.95
Average traffic	41.11
Heavy traffic	34.55

It is necessary to emphasize that those results are obtained in a small section of the city, therefore they could increase as much as the analyzed scenario is larger. The average time of travel of the vehicles obtained was in an extension of 700 m of length of Avenida Paulista. The total length of this avenue is approximately 4 km. Therefore, if we increase the size of the analyzed scenario we would obtain larger differences between the average travel times with fixed time semaphore control and the proposed BiNN algorithm. The same could happen with the level of occupation of the lanes, which we are analyzing only in an area of 4.4 km of streets and avenues, compared to more than 17,000 km of roads that have the city of São Paulo.

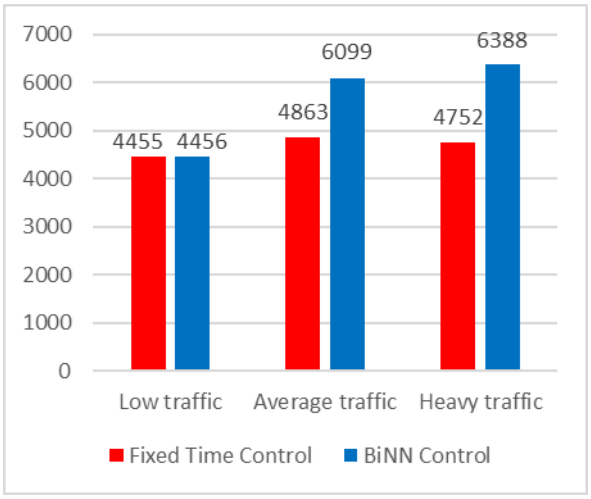


Fig 9. Number of vehicles departed

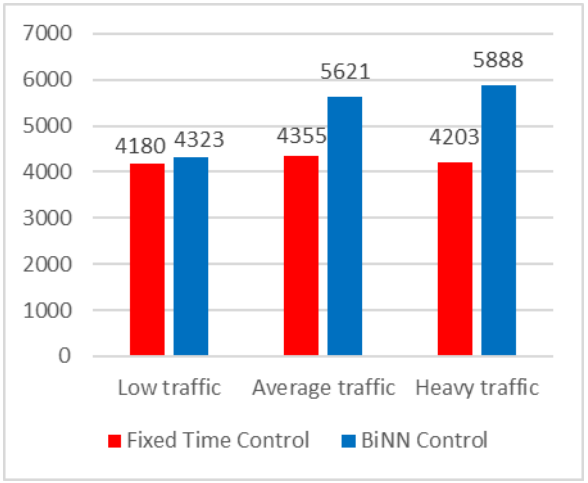


Fig 10. Number of vehicles arrived

VI. CONCLUSIONS

This work describes the application of an algorithm based on Bio-Inspired Neural Networks for the multi-agent control of traffic semaphores, with the purpose of controlling urban traffic and reducing the level of traffic bottling. The algorithm was applied in a real scenario in the city center region of São Paulo obtaining good results, which were compared with results obtained with fixed times control. Test results show up to 22.89% performance under the low traffic conditions, 26.35% under average traffic and 28.86% under the heavy traffic condition compared to the fixed time control, the performance indicators used for this comparison were the average travel time and the level of occupation of lanes. One of the main contributions of this paper is the ability to estimate the real impact on the traffic level that BiNN control could apply in the city of São Paulo. For further research, it is possible to analyze the performance indicators used in this work with increasing of the size of the scenario, the length of the vials and the number of traffic semaphores.

ACKNOWLEDGMENT

This work was supported by CAPES, *Coordenação de Aperfeiçoamento de PEssoal de nível Superior*. Also we would thanks to the *Institute of Transportation Systems* (DLR) for allow the use of SUMO simulation software.

REFERENCES

[1] Department of Economic and Social Affairs of the United Nations Secretariat (DESA), *World population prospects. Key finding and advance tables*, vol. 28, no. 3. 2016.

[2] A. Arroub, B. Zahi, E. Sabir, and M. Sadik, "A literature review on Smart Cities: Paradigms, opportunities and open problems," *2016 Int. Conf. Wirel. Networks Mob. Commun.*, pp. 180–186, 2016.

[3] Y. Hernafi, M. Ben Ahmed, and M. Bouhorma, "An Approaches ' based on Intelligent Transportation Systems to Dissect Driver Behavior and Smart Mobility in Smart City," pp. 886–895, 2016.

[4] F. Vit, N. Mirko, S. Miroslav, and V. Zdenek, "System alliances as a tool for solving Smart Cities problems," *2015 Smart Cities Symp. Prague, SCSP 2015*, 2015.

[5] I. S. and S. SIEMENS, "Solutions for Urban Traffic, Intelligent Traffic Systems."

[6] J. Von Stritzky and C. Cabrerizo, "Ideas paras las ciudades inteligentes del futuro," p. 64, 2011.

[7] G. B. Castro, J. S. C. Martini, and A. R. Hirakawa, "Biologically-Inspired Neural Network for Traffic Signal Control," *IEEE 17th Int. Conf. Intell. Transp. Syst.*, pp. 2144–2149, 2014.

[8] G. B. Castro, D. S. Miguel, B. P. Machado, and A. R. Hirakawa, "Biologically-inspired Neural Network for Coordinated Urban Traffic Control Parameter Determination and Stability Analysis," *Int. Conf. Comput. Sci. Comput. Intell.*, pp. 209–214, 2015.

[9] A. Diveev, E. Sofronova, and V. Mikhalev, "Model Predictive Control for Urban Traffic Flows," *IEEE Int. Conf. Syst. Man, Cybern.*, pp. 3051–3056, 2016.

[10] S. Araghi, A. Khosravi, and D. Creighton, "Optimal Design of Traffic Signal Controller Using Neural Networks and Fuzzy Logic Systems," *Int. Jt. Conf. Neural Networks*, pp. 42–47, 2014.

[11] M. B. W. de Oliveira and A. de A. Neto, "Optimization of Traffic Lights Timing based on Artificial Neural," *IEEE 17th Int. Conf. Intell. Transp. Syst.*, pp. 1921–1922, 2014.

[12] M. Elgarej, M. Khalifa, and M. Youssfi, "Traffic Lights Optimization with Distributed Ant Colony Optimization Based on Multi-agent System," *Springer Int. Publ. AG*, pp. 266–279, 2016.

[13] T. Kaur and S. Agrawal, "Adaptive Traffic Lights Based On Hybrid of Neural Network and Genetic Algorithm for Reduced Traffic Congestion," *Eng. Comput. Sci. (RAECS), 2014 Recent Adv.*, pp. 266–279, 2014.

[14] G. B. Castro, "Modelo de rede neural bioinspirada para o controle do trânsito urbano," Universidade de São Paulo, 2016.

[15] J. R. Peláez and D. Andina, "Do biological synapses perform probabilistic computations?," *Neurocomputing*, vol. 114, pp. 24–31, 2013.

[16] "Demographia World Urban Areas," *Demographia*. p. 112, 2016.

[17] Prefeitura de São Paulo, "Companhia de Engenharia de Tráfego de São Paulo," 2017. [Online]. Available: http://www.cetsp.com.br/. [Accessed: 02-Apr-2017].

[18] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "SUMO – Simulation of Urban MObility An Overview," *Inst. Transp. Syst.*, pp. 1–6, 2011.

[19] Robert Hilbrich, "SUMO – Simulation of Urban MObility," *Institute of Transportation System*. [Online]. Available: http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/. [Accessed: 07-Apr-2017].

[20] A. F. Acosta Gil, "TraCI4Matlab: User's Manual," 1, 2014.

[21] Globo Comunicação e Participações S.A, "Radar do trânsito em tempo real de São Paulo," 2017. [Online]. Available: http://g1.globo.com/sao-paulo/transito/radar-tempo-transito-agora.html.

[22] B. L. Ye, W. Wu, L. Li, and W. Mao, "A hierarchical model predictive control approach for signal splits optimization in large-scale urban road networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 8, pp. 2182–2192, 2016.

Parte II

Transcrição da mesa redonda

A transcrição da mesa redonda está em fase de revisão. Em breve, esta seção será substituída pelo texto adequado.