COMPUTATION TOOLS 2024

The Fifteenth International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking

ISBN: 978-1-68558-158-9

April 14 - 18, 2024

Venice, Italy

COMPUTATION TOOLS 2024 Editors

Claus-Peter Rückemann, Universität Münster / DIMF / Leibniz Universität Hannover, Germany

COMPUTATION TOOLS 2024

Forward

The Fifteenth International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking (COMPUTATION TOOLS 2024), held between April 14 – 18, 2024, continued a series of events dealing with logics, algebras, advanced computation techniques, specialized programming languages, and tools for distributed computation. Mainly, the event targeted those aspects supporting context-oriented systems, adaptive systems, service computing, patterns and content-oriented features, temporal and ubiquitous aspects, and many facets of computational benchmarking.

Similar to the previous edition, this event attracted excellent contributions and active participation from all over the world. We were very pleased to receive top quality contributions.

We take here the opportunity to warmly thank all the members of the COMPUTATION TOOLS 2024 technical program committee, as well as the numerous reviewers. The creation of quality conference program would not have been possible without their involvement. We also kindly thank all the authors that dedicated much of their time and effort to contribute to COMPUTATION TOOLS 2024. We truly believe that, thanks to all these efforts, the final conference program consisted of top quality contributions.

Also, this event could not have been a reality without the support of many individuals, organizations and sponsors. We also gratefully thank the members of the COMPUTATION TOOLS 2024 organizing committee for their help in handling the logistics and for their work that made this professional meeting a success.

We hope COMPUTATION TOOLS 2024 was a successful international forum for the exchange of ideas and results between academia and industry and to promote further progress in the area of computational logics, algebras, programming, tools, and benchmarking. We also hope that Venice provided a pleasant environment during the conference and everyone saved some time to enjoy this beautiful city.

COMPUTATION TOOLS 2024 Steering Committee

Cornel Klein, Siemens AG, Germany
Claus-Peter Rückemann, Westfälische Wilhelms-Universität Münster (WWU) / DIMF / Leibniz Universität Hannover, Germany

COMPUTATIONAL TOOLS 2024 Publicity Chairs

José Miguel Jiménez, Universitat Politecnica de Valencia, Spain
Sandra Viciano Tudela, Universitat Politecnica de Valencia, Spain

COMPUTATION TOOLS 2024

Committee

COMPUTATION TOOLS 2024 Steering Committee

Cornel Klein, Siemens AG, Germany
Claus-Peter Rückemann, Westfälische Wilhelms-Universität Münster (WWU) / DIMF / Leibniz Universität Hannover, Germany

COMPUTATIONAL TOOLS 2024 Publicity Chairs

José Miguel Jiménez, Universitat Politecnica de Valencia, Spain
Sandra Viciano Tudela, Universitat Politecnica de Valencia, Spain

COMPUTATION TOOLS 2024 Technical Program Committee

Lorenzo Bettini, Università di Firenze, Italy
Ateet Bhalla, Independent Consultant, India
Narhimene Boustia, University Saad Dahlab, Blida 1, Algeria
Azahara Camacho, Opinno, Spain
Angelo Ciaramella, University of Naples Parthenope, Italy
Cornel Klein, Siemens AG, Germany
Emanuele Covino, Universita' di Bari, Italy
Marcos Cramer, TU Dresden, Germany
Santiago Escobar, VRAIN - Universitat Politècnica de València, Spain
Andreas Fischer, Deggendorf Institute of Technology, Germany
Shengzhong Mao, University of Manchester, UK
Roderick Melnik, Wilfrid Laurier University, Canada
Corrado Mencar, Università degli Studi di Bari Aldo Moro, Italy
Ralph Müller-Pfefferkorn, Technische Universität Dresden, Germany
Keiko Nakata, SAP SE - Potsdam, Germany
Adam Naumowicz, University of Bialystok, Poland
Cecilia E. Nugraheni, Parahyangan Catholic University, Indonesia
Alberto Policriti, University of Udine, Italy
Kristin Yvonne Rozier, Iowa State University, USA
James Tan, Singapore University of Social Sciences, Singapore
Hans Tompits, Technische Universität Wien, Austria
Prajna Upadhyay, BITS Pilani Hyderabad, India
Miroslav Velev, Aries Design Automation, USA
Tao Zheng, Orange Innovation China, China

Copyright Information

For your reference, this is the text governing the copyright release for material published by IARIA.

The copyright release is a transfer of publication rights, which allows IARIA and its partners to drive the dissemination of the published material. This allows IARIA to give articles increased visibility via distribution, inclusion in libraries, and arrangements for submission to indexes.

I, the undersigned, declare that the article is original, and that I represent the authors of this article in the copyright release matters. If this work has been done as work-for-hire, I have obtained all necessary clearances to execute a copyright release. I hereby irrevocably transfer exclusive copyright for this material to IARIA. I give IARIA permission or reproduce the work in any media format such as, but not limited to, print, digital, or electronic. I give IARIA permission to distribute the materials without restriction to any institutions or individuals. I give IARIA permission to submit the work for inclusion in article repositories as IARIA sees fit.

I, the undersigned, declare that to the best of my knowledge, the article is does not contain libelous or otherwise unlawful contents or invading the right of privacy or infringing on a proprietary right.

Following the copyright release, any circulated version of the article must bear the copyright notice and any header and footer information that IARIA applies to the published article.

IARIA grants royalty-free permission to the authors to disseminate the work, under the above provisions, for any academic, commercial, or industrial use. IARIA grants royalty-free permission to any individuals or institutions to make the article available electronically, online, or in print.

IARIA acknowledges that rights to any algorithm, process, procedure, apparatus, or articles of manufacture remain with the authors and their employers.

I, the undersigned, understand that IARIA will not be liable, in contract, tort (including, without limitation, negligence), pre-contract or other representations (other than fraudulent misrepresentations) or otherwise in connection with the publication of my work.

Exception to the above is made for work-for-hire performed while employed by the government. In that case, copyright to the material remains with the said government. The rightful owners (authors and government entity) grant unlimited and unrestricted permission to IARIA, IARIA's contractors, and IARIA's partners to further distribute the work.

Table of Contents

A Note on a Syntactical Measure of the Complexity of Programs

Emanuele Covino
Dipartimento di Informatica
Universitá degli Studi di Bari
Bari, Italy
emanuele.covino@uniba.it

***Abstract*—We introduce a programming language operating on stacks and a syntactical measure σ, such that a natural number $\sigma(\mathsf{P})$ is assigned to each program P. The measure considers how the presence of loops defined over size-increasing (and non-size-increasing) subprograms influences the complexity of the program itself. Functions computed by a program of σ-measure n are exactly those computable by a Turing machine with running time in $\mathcal{E}^{n+2}$ (the $n+2$-th Grzegorczyk class). Programs of σ-measure 0 compute the polynomial-time computable functions. Thus, we have a syntactical characterization of the functions belonging to the Grzegorczyk hierarchy; this result represents an improvement with respect to previous similar results.**

***Index Terms*—polynomial-time complexity, Grzegorczyk hierarchy, imperative programming languages, stack programs**

I. Introduction

The definition of a class of functions with a given computational complexity is usually given by introducing an explicit bound on time and/or space resources used by a Turing Machine during the computation of the functions. Other approaches capture complexity classes by means of some form of *limited recursion*; the first characterization of this type has been given by Cobham [3], who has shown that the polynomial-time computable functions are exactly those that are definable by *bounded recursion on notation*, starting from a set of simple basic functions.

In the recent years, a number of characterizations of complexity classes has been given, showing that they can be captured by means of various forms of *ramified recursion*, without any explicitly bounded scheme of recursion. Initiated by Simmons [23], Bellantoni and Cook [1] and Leivant [13] - [15], one can find functional characterization of linear-space/time computable functions LINSPACE and LOGSPACE [9], polynomial time [16], polynomial space [18] [21], the elementary functions [21] [4], non-size-increasing computations [6], among the others.

A different approach can be found in [7] [8] [10] [11]; more recently, in [12] [17]. The properties of *imperative* programs (such as complexity, resource utilization, termination) are now investigated by analyzing their syntax only. In particular, the properties of a programming language operating on stacks are studied in [10]; the language supports loops over stacks, conditionals and concatenation, besides the usual pop and push operations (see Section II for the detailed semantics). The natural concept of μ-measure is then introduced; it is a syntactical method by which one is able to assign to each program P a number $\mu(\mathsf{P})$. It is proved the following *bounding theorem*: functions computed by stack programs of μ measure n have a length bound $b \in \mathcal{E}^{n+2}$ (the $n+2$-th Grzegorczyk class), that is $|f(\vec{w})| \leq b(|\vec{w}|)$; as a consequence, stack programs of measure 0 have polynomial running time, and programs of measure n compute functions whose time complexity is in the $n+2$-th finite level of the Grzegorczyk hierarchy. This result provides a measure of the impact of nesting loops on computational complexity; if a stack Z is updated into a loop controlled by a stack Y and, afterwards, Y is updated into a loop controlled by Z, we have a so called *top circle* in the program; when this circular reference occurs into an external loop, a blow up in the complexity of the program is produced. The μ-measure is a syntactical way to detect top circles; each time one of them occurs in the body of a loop, the μ measure is increased by 1 (see below, Section III and definition 3.1).

There are various ways of improving the measure μ (for instance, see [11]), since it is an undecidable problem whether or not a function computed by a given stack program lies in a given complexity class. In this paper, we provide a refinement of μ, starting from the following consideration: a program whose structure leads the μ-measure to be equal to n contains n nested top circles, and this implies, by the bounding theorem, that the program has a length bound $b \in \mathcal{E}^{n+2}$. Suppose now that some of the sequences of pop and push (or, in general, some of the subprograms) iterated into the main program leave unchanged the overall space used; since not increasing programs can be iterated without leading to any growth in space, the effective space bound is lower than the bound obtained via the μ-measure, and it can be evaluated by an alternative measure σ. While μ grows each time a top circle appears in the body of a loop, σ grows only for *increasing* top circles. In other words, the new measure doesn't consider those situations in which some (potentially harmful) operations are performed, but

 ISBN: 978-1-68558-158-9

their overall balance is negative. We prove a new bounding theorem using the σ-measure, providing a more appropriate bound to the complexity of stacks programs.

In Section II, we recall concepts and definitions of stack programs and of the Grzegorczyk hierarchy. In Section III, we recall the definition of μ-measure. In Section IV, we introduce the definition of the new σ-measure and the new bounding theorem. Conclusions and future work can be found in Section V.

II. Preliminaries on the Grzegorczyk hierarchy and on stack programs

In this section, we recall the definition of the Grzegorczyk hierarchy, and fundamentals on stack programs and their computations; the reader is referred to [22] [5] [2] [10] for complete definitions and properties.

Definition 2.1: Given a unary function f, the k-th *iterate* of f (denoted with f^k) is $f^0(x) = x$ and $f^{k+1}(x) = f(f^k(x))$.

Definition 2.2: The *principal functions* $E_1, E_2, E_3, \ldots$ are $E_1(x) = x^2 + 2$ and $E_{n+2}(x) = E^x_{n+1}(2)$ (the x-th iterate of E_{n+1}).

Definition 2.3: A function f is defined by *bounded recursion* from functions g, h, and b if for all $\vec{x}$, y one has $f(\vec{x}, 0) = g(\vec{x})$, $f(\vec{x}, y) = h(\vec{x}, y, f(\vec{x}))$ and $f(\vec{x}, y) \leq g(\vec{x}, y)$.

Definition 2.4: The n-th Grzegorczyk class $\mathcal{E}^n$ is the least class of functions containing the initial functions zero, successor, projections, maximum and E_{n-1}, and closed under composition and bounded recursion.

Stack programs operate on variables serving as stacks; they contain arbitrary words over a fixed alphabet Σ, and they are manipulated by running a program built from imperatives push(a,X), pop(X), and nil(X) combined by sequencing, conditional, and loop statements (respectively, P;Q, if top(X)≡a then [P], and foreach X [P]).

Definition 2.5: The operational semantics of stack programs are defined as follows:

1) push(a,X) pushes a letter a on the top of the stack X;
2) pop(X) removes the top of X, if any; it leaves X unchanged, otherwise;
3) nil(X) empties the stack X;
4) if top(X)≡a [P] executes P if the top of the stack X is equal to a;
5) $P_1;\ldots;P_k$ are executed from left to right;
6) foreach X [P] executes P for |X| times

with the restriction that no imperatives over X may occur in the body of a loop foreach X [P] (i.e., in P), and that the loop is executed call-by-value; X is the *control stack* of the loop. |X| stands for the length of the word stored in X.

The notation $\{A\}\mathsf{P}\{B\}$ means that if the condition expressed by the sentence A holds before the execution of P, then the condition expressed by the sentence B holds after the execution of P.

Definition 2.6: A stack program P *computes* a function $f : (\Sigma^*)^n \to \Sigma^*$ if P has an output variable O and n input variables $\bar{\mathsf{X}} = \mathsf{X}_{i_1}, \ldots, \mathsf{X}_{i_n}$ among stacks $\mathsf{X}_1, \ldots, \mathsf{X}_m$ such that $\{\bar{\mathsf{X}} = \vec{w}\}\mathsf{P}\{\mathsf{O} = f(\vec{w})\}$, for all $\vec{w} = w_1, \ldots, w_n \in (\Sigma^*)^n$.

For a fixed program P, two sets of variables are defined: $\mathcal{U}(\mathsf{P}) = \{\mathsf{X} | \mathsf{P}$ contains an imperative push(a,X)$\}$ and $\mathcal{C}(\mathsf{P}) = \{\mathsf{X} | \mathsf{P}$ contains a loop foreach X [Q], and $\mathcal{U}(\mathsf{Q}) \neq \emptyset\}$. Variables in $\mathcal{U}(\mathsf{P})$ can be altered by a push during a run of P, while variables in $\mathcal{C}(\mathsf{P})$ control a loop occurring in P. The two sets are not disjoint.

Definition 2.7: X *controls* Y in the program P (denoted with $\mathsf{X} \prec_{\mathsf{P}} \mathsf{Y}$) if P contains a loop foreach X [Q], and $\mathsf{Y} \in \mathcal{U}(\mathsf{Q})$; the transitive closure of $\prec_{\mathsf{P}}$ is denoted by $\overset{\mathsf{P}}{\to}$.

Consider the following program:

P_1:= foreach X_1[... foreach X_l [push (a,Y)]]

If words $v_1 \ldots v_l, w$ are stored in $\mathsf{X}_1 \ldots \mathsf{X}_l$, Y, respectively, before P_1 is executed, then Y holds the word $w\mathsf{a}^{|v_1|\ldots|v_l|}$ after the execution of P_1. The depth of loop-nesting is a necessary condition for high computational complexity, but it is not a sufficient condition. Now, consider the following two programs:

```
P2:= nil(Y); push(a,Y); nil(Z); push(a,Z);
     foreach X [nil(Z); foreach Y [push(a,Z); push(a,Z)];
                nil(Y); foreach Z [push(a,Y)]]

P3:= nil(Y); push(a,Y); nil(Z);
     foreach X [
          foreach Y [push(a,Z); push(a,Z); push(a,Y)]]
```

Even if both P_2 and P_3 have nesting depth 2, if w is initially stored in X, then Z holds the word $\mathsf{a}^{2^{|w|}}$ after P_2 is executed, while $\mathsf{a}^{|w|(|w|+1)}$ is stored in Z after the execution of P_3. Thus, we see that P_3 runs in polynomial time, whereas P_2 has exponential running time. This happens because of the (control) circle contained inside the outermost loop in P_2: inside the loop governed by X, first Y *controls* Z (in that Z is updated via push(a,Z) inside a loop governed by Y), and then Z *controls* Y in the same sense. In contrast, there is no such circle in P_3. Stack programs where each body of a loop statement is circle-free compute exactly the functions computable within polynomial time, and must be separated from those programs with loops that cause a blow up in running time.

III. The μ-measure on stack programs

Starting from the previous relation $\overset{\mathsf{P}}{\to}$, a measure over the set of stack programs is introduced in [10].

Definition 3.1: Let P be a stack program. The *μ-measure of* P (denoted with $\mu(\mathsf{P})$) is defined as follows, inductively:

1) $\mu(\mathsf{pop}) = \mu(\mathsf{push}) = \mu(\mathsf{nil}) := 0$;
2) $\mu(\mathsf{if\ top(X)}{\equiv}\mathsf{a\ [Q]}) := \mu(\mathsf{Q})$;

3) $\mu(\mathsf{P};\mathsf{Q}) := \max(\mu(\mathsf{P}), \mu(\mathsf{Q}))$;
4) $\mu(\mathsf{foreach\ X\ [Q]}) := \mu(\mathsf{Q}) + 1$, if Q is a sequence $\mathsf{Q}_1;\ldots;\mathsf{Q}_l$ with a *top circle* (that is, if there exists Q_i such that $\mu(\mathsf{Q}_i) = \mu(\mathsf{Q})$, some Y controls some Z in Q_i, and Z controls Y in $\mathsf{Q}_1;\ldots;\mathsf{Q}_{i-1};\mathsf{Q}_{i+1};\ldots;\mathsf{Q}_l$); $\mu(\mathsf{foreach\ X\ [Q]}) := \mu(\mathsf{Q})$, otherwise.

To focus on the critical case where P is a loop foreach X [Q], assume that μ(Q) is already determined. Suppose that Q is a sequence $\mathsf{Q}_1;\ldots;\mathsf{Q}_l$, in which case μ(Q) is max(Q_1, ...,Q_l). Then a blow up in running time can only occur if Q has a top circle, that is, Q has a circle with respect to a control variable Y of some component Q_i of maximal μ-measure μ(Q). In this case, μ(P) is defined as μ(Q)+1. In all other cases, μ(P) is defined as μ(Q). Given that all primitive instructions receive μ-measure 0, one easily verifies for the examples above that $\mu(\mathsf{P}_1)$=$\mu(\mathsf{P}_3)$=0, whereas $\mu(\mathsf{P}_2)$=1.

The core of [10] is the following bounding theorem.

Lemma 3.1: Every function f computed by a stack program of μ-measure n has length bound $b \in \mathcal{E}^{n+2}$ satisfying $|f(\vec{w})| \leq b(|\vec{w}|)$, for all $\vec{w}$. In particular, if P computes a function f, and $\mu(\mathsf{P}) = 0$, then f has a polynomial length bound, that is, there exists a polynomial p satisfying $|f(\vec{w})| \leq p(|\vec{w}|)$.

Let $\mathcal{L}^n_\mu$ be the class of all functions which can be computed by a stack program of μ-measure $n \geq 0$, and let $\mathcal{G}^n$ be the class of all functions which can be computed by a Turing machine in time $b(|\vec{w}|)$, for some $b \in \mathcal{E}^n$. As a consequence of the bounding lemma, the following result holds.

Theorem 3.1: For $n \geq 0$*:* $\mathcal{L}^n_\mu = \mathcal{G}^{n+2}$.

IV. The σ-measure and a new bounding theorem

In the rest of the paper, we denote with imp(Y) an imperative pop(Y), push(a,Y), or nil(Y); we denote with mod($\bar{\mathsf{X}}$) a *modifier*, that is a sequence of imperatives operating on the variables occurring in $\bar{\mathsf{X}} = \mathsf{X}_1, \ldots, \mathsf{X}_n$. We introduce a modified definition of *circle*, which better matches our new measure.

Definition 4.1: Let Q be a sequence in the form $\mathsf{Q}_1;\ldots;\mathsf{Q}_l$. There is a *circle* in Q if there exists a sequence of variables Z_1,Z_2,...,Z_l, and a permutation π of $\{1,\ldots,l\}$ such that $\mathsf{Z}_1 \overset{\mathsf{Q}_{\pi(1)}}{\rightarrow} \mathsf{Z}_2 \overset{\mathsf{Q}_{\pi(2)}}{\rightarrow} \ldots \mathsf{Z}_l \overset{\mathsf{Q}_{\pi(l)}}{\rightarrow} \mathsf{Z}_1$. The subprograms $\mathsf{Q}_1,\ldots,\mathsf{Q}_l$ and the variables $\mathsf{Z}_1,\ldots,\mathsf{Z}_l$ are *involved* in the circle.

For sake of simplicity, we will consider $\pi(i) = i$, that is the case $\mathsf{Z}_1 \overset{\mathsf{Q}_1}{\rightarrow} \mathsf{Z}_2 \overset{\mathsf{Q}_2}{\rightarrow} \ldots \mathsf{Z}_l \overset{\mathsf{Q}_l}{\rightarrow} \mathsf{Z}_1$; proofs and definitions holds in the general case too.

Definition 4.2: Let P be a stack program and let Y be a given variable. The *σ-measure of* P *with respect to* Y (denoted with σ_Y(P)) is defined as follows, inductively (with $sg(z) = 1$ if $z \geq 1$, $sg(z) = 0$ otherwise):

1) $\sigma_\mathsf{Y}(\mathsf{mod}(\bar{\mathsf{X}})) := sg(\sum \hat{\sigma}_\mathsf{Y}(\mathsf{imp(Y)}))$, for each imp(Y) $\in$ mod($\bar{\mathsf{X}}$), where
   $\hat{\sigma}_\mathsf{Y}(\mathsf{push(a,Y)}) := 1$;
   $\hat{\sigma}_\mathsf{Y}(\mathsf{pop(Y)}) := -1$;
   $\hat{\sigma}_\mathsf{Y}(\mathsf{nil(Y)}) := -\infty$;
   $\hat{\sigma}_\mathsf{Y}(\mathsf{imp(X)}) := 0$, with Y$\neq$X;
2) $\sigma_\mathsf{Y}(\mathsf{if\ top\ Z} \equiv \mathsf{a[P]}) := \sigma_\mathsf{Y}(\mathsf{P})$;
3) $\sigma_\mathsf{Y}(\mathsf{P}_1;\mathsf{P}_2) := \max(\sigma_\mathsf{Y}(\mathsf{P}_1), \sigma_\mathsf{Y}(\mathsf{P}_2))$, with $\mathsf{P}_1;\mathsf{P}_2$ not a modifier;
4) $\sigma_\mathsf{Y}(\mathsf{foreach\ X\ [Q]}) := \sigma_\mathsf{Y}(\mathsf{Q})+1$, if there exists a circle in Q, and a subprogram Q_i s.t.
   (a) Y and Q_i are involved in the circle;
   (b) $\sigma_\mathsf{Y}(\mathsf{Q}) = \sigma_\mathsf{Y}(\mathsf{Q}_i)$;
   (c) the circle is increasing;
   $\sigma_\mathsf{Y}(\mathsf{foreach\ X\ [Q]}) := \sigma_\mathsf{Y}(\mathsf{Q})$, otherwise,

where a circle is *not increasing* if, denoted with Q_1,Q_2,...,Q_l and with Z_1,Z_2,...,Z_l the sequences of subprograms and, respectively, of variables involved in the circle, we have that $\sigma_{\mathsf{Z}_i}(\mathsf{Q}_j) = 0$, for each $i := 1\ldots l$ and $j := 1\ldots l$. If the previous condition doesn't hold, we say that the circle is *increasing*.

Note that the σ_Y-measure of a modifier (see (1) in the previous definition) is equal to 1 only when, in absence of nil's, the overall number of push's over Y is greater than the number of pop's over the same variable, that is, only when a growth in the length of Y is produced. Moreover, note that the "otherwise" case in (4) can be split in three different cases. First, there are no circles in which Y is involved. Second, Y is involved, together with a subprogram Q_i, in a circle in Q, but it happens that $\sigma_\mathsf{Y}(\mathsf{Q}_i)$ is lower than σ_Y(Q) (this means that there is a blow-up in the complexity of Y in $\sigma_\mathsf{Y}(\mathsf{Q}_i)$, but this growth is still bounded by the complexity of Y in a different subprogram of Q). Third, Y is involved in some circles in Q, but each of them is not increasing (that is, according to the previous definition, each variable Z_i involved in each circle doesn't produce a growth in the complexity of the subprograms Q_j involved in the same circle). This implies that the space used during the execution of the external loop foreach X [Q] is basically the same used by Q (this is not a surprising fact: one can freely iterate a not increasing program without leading an harmful growth). In all three cases the σ-measure must remain unchanged: it is increased only when an increasing top circle occurs and when at least one of the variables involved in that circle causes a growth in the space complexity of the related subprogram, simultaneously (that is, if there exists a p such that $\sigma_{\mathsf{Z}_p}(\mathsf{Q}_p) > 0$).

In the following definition, we extend the measure to the whole set of variables occurring in a stack program.

Definition 4.3: Let P be a stack program. The *σ-measure of* P is $\sigma(\mathsf{P}) := \tilde{\sigma}(\mathsf{P}) \dot{-} 1$, where $\dot{-}$ is the usual cut-off subtraction, and

1) $\tilde{\sigma}(\mathsf{mod}(\bar{\mathsf{X}})) := 0$
2) $\tilde{\sigma}(\mathsf{if\ top\ Z} \equiv \mathsf{a\ [Q]}) := \max(\sigma_\mathsf{Y}(\mathsf{if\ top\ Z} \equiv \mathsf{a\ [Q]}))$, for all Y occurring in P;
3) $\tilde{\sigma}(\mathsf{P}_1;\mathsf{P}_2) := \max(\sigma_\mathsf{Y}(\mathsf{P}_1;\mathsf{P}_2))$, for all Y occurring in P, with $\mathsf{P}_1;\mathsf{P}_2$ not a modifier;

 ISBN: 978-1-68558-158-9

4) $\tilde{\sigma}(\mathsf{foreach\ X\ [Q]}) := \max(\sigma_{\mathsf{Y}}(\mathsf{foreach\ X\ [Q]}))$, for all Y occurring in P.

Note that $\sigma(\mathsf{P}) \leq \mu(\mathsf{P})$, for each stack program P. Note also that we are using the previously defined $\hat{\sigma}_{\mathsf{Y}}$ to detect all the *increasing* modifiers, for a given variable Y (this is done setting $\hat{\sigma}_{\mathsf{Y}}$ equal to 1); but, once detected, we don't have to consider them in the evaluation of the σ-measure. This is the reason of the "$\dot{-}1$" part in the previous definition.

In the rest of the paper we introduce a reduction procedure between stack programs, denoted with $\rightsquigarrow$, and we prove a new bounding theorem.

Definition 4.4: P and Q are *space equivalent* if $\{\bar{\mathsf{X}} = \vec{w}\}\mathsf{P}\{|\bar{\mathsf{X}}| = m\}$ implies that $\{\bar{\mathsf{X}} = \vec{w}\}\mathsf{Q}\{|\bar{\mathsf{X}}| = O(m)\}$. This is denoted with $\mathsf{P} \approx_s \mathsf{Q}$.

Definition 4.5: Let A be a stack program such that $\mu(\mathsf{A}) = n+1$, and $\sigma(\mathsf{A}) = m$, with $m < n+1$; the program $\rightsquigarrow\mathsf{A}$ is obtained as follows:

1) if A is foreach X [R], with $\mu(\mathsf{R}) = \sigma(\mathsf{R}) = n$, and denoted with $C_1, \ldots, C_l$ the top circles in R, and with $\mathsf{A}_{i1}, \ldots, \mathsf{A}_{ip}$ the variables involved in C_i, for each i, we have that $\rightsquigarrow\mathsf{A}$ is the result of changing each $\mathsf{imp}(\mathsf{A}_{ij})$ into $\mathsf{nop}(\mathsf{A}_{ij})$ (a *no-operation* imperative);
2) if A is foreach X [R], with $\mu(\mathsf{R}) > \sigma(\mathsf{R})$, , we have that $\rightsquigarrow\mathsf{A}$ is equal to foreach X [$\rightsquigarrow$R];
3) if A is $\mathsf{A}_1;\mathsf{A}_2$ and $\max(\mu(\mathsf{A}_1), \mu(\mathsf{A}_2)) = \mu(\mathsf{A}_1)$, we have that $\rightsquigarrow\mathsf{A}$ is equal to $\rightsquigarrow\mathsf{A}_1;\mathsf{A}_2$;
   simmetrically, if $\max(\mu(\mathsf{A}_1), \mu(\mathsf{A}_2)) = \mu(\mathsf{A}_2)$, we have that $\rightsquigarrow\mathsf{A}$ is equal to $\mathsf{A}_1;\rightsquigarrow\mathsf{A}_2$;
   if $\mu(\mathsf{A}_1) = \mu(\mathsf{A}_2)$, we have that $\rightsquigarrow\mathsf{A}$ is equal to $\rightsquigarrow\mathsf{A}_1;\rightsquigarrow\mathsf{A}_2$;
4) if A is if top(X)≡a [R], we have that $\rightsquigarrow\mathsf{A}$ is equal to if top(X)≡a [$\rightsquigarrow$R].

Lemma 4.1: Given a stack program P, with $\mu(\mathsf{P}) = n+1$ and $\sigma(\mathsf{P}) = n$, there exists a stack program $\rightsquigarrow\mathsf{P}$ such that $\mu(\rightsquigarrow \mathsf{P}) = n$, $\sigma(\rightsquigarrow \mathsf{P}) = n$, and $\mathsf{P} \approx_s \rightsquigarrow\mathsf{P}$.

Proof. (by induction on n). Base. Let $\mu(\mathsf{P}) = 1$ and $\sigma(\mathsf{P}) = 0$. In the main case, P is in the form foreach X [Q], with a not-incresing circle occurring in Q. Applying $\rightsquigarrow$ to P, we obtain a program P' whose σ-measure is still 0, and whose μ-measure is reduced to 0, because $\rightsquigarrow$ has broken off the circle in P that leads μ from 0 to 1 (i.e., in P', there are no more push's on the variables involved in the circle). Note that P can decrease the length of the stacks involved in the circle, while P' does not perform any operation in the same circle. Thus, P' can increase its variables only by a linear factor; indeed, if $\{\bar{\mathsf{X}} = \vec{w}\}\mathsf{P}\{|\bar{\mathsf{X}}| = m\}$ we have that $\{\bar{\mathsf{X}} = \vec{w}\}\mathsf{P}'\{|\bar{\mathsf{X}}| = cm\}$, where c is a constant depending on the structure of P: thus, $\mathsf{P} \approx_s \mathsf{P}'$.

Step. Let $\mu(\mathsf{P}) = n+2$ and $\sigma(\mathsf{P}) = n+1$. Let P be in the form foreach X [Q], and let C be a top circle occurring in Q, with $\mu(\mathsf{Q}) = n+1$; we have two cases: (1) $\sigma(\mathsf{Q}) = n+1$, or (2) $\sigma(\mathsf{Q}) = n$.

(1) In this case C is a not-increasing circle, because it has been detected by μ, but not by σ. Applying $\rightsquigarrow$ to P, we obtain a program P' such that $\sigma(\mathsf{P}') = n+1$, $\mu(\mathsf{P}') = n+1$, and $\mathsf{P} \approx_s \mathsf{P}'$.

(2) In this case C is an increasing circle, detected by μ and σ. We have that (by the inductive hypothesis) there exists a program Q' such that $\mu(\mathsf{Q}') = n$, $\sigma(\mathsf{Q}') = n$, and $\mathsf{Q} \approx_s \mathsf{Q}'$. Starting from P, we build a new program P'=foreach X [Q'] . We have that $\mu(\mathsf{P}') = \mu(\mathsf{Q}') + 1 = n+1$, $\sigma(\mathsf{P}') = \sigma(\mathsf{Q}') + 1 = n+1$, and $\mathsf{P} \approx_s \mathsf{P}'$ as expected.

The cases $\mathsf{P}_1;\mathsf{P}_2;\ldots;\mathsf{P}_k$ and if top(X)≡a [P] can be proved in a similar way.

Theorem 4.1: Every function f computed by a stack program P such that $\mu(\mathsf{P}) = n$ and $\sigma(\mathsf{P}) = m$, with $n > m$, has a length bound $b \in \mathcal{E}^{m+2}$ satisfying $|f(\vec{w})| \leq b(|\vec{w}|)$.

Proof. Let k be $\mu(\mathsf{P}) - \sigma(\mathsf{P})$. Then by k applications of Lemma 4.1, we obtain a sequence $\mathsf{P} =: \mathsf{P}_0, \mathsf{P}_1, \ldots, \mathsf{P}_k$ of stack programs such that, for all $i < k$,

$$\mu(\mathsf{P}_{i+1}) = \mu(\mathsf{P}) - i,\ \sigma(\mathsf{P}_i) = \sigma(\mathsf{P}_{i+1}), \text{ and } \mathsf{P}_i \approx_s \mathsf{P}_{i+1}.$$

By Kristiansen and Niggl's bounding theorem, P_k has a length bound in $\mathcal{E}^{\sigma(\mathsf{P})+2}$, and so does P, by transitivity of $\approx_s$.

Let $\mathcal{L}^n_\sigma$ be the class of all functions that can be computed by a stack program of σ-measure $n \geq 0$, and let $\mathcal{G}^n$ be the class of all functions which can be computed by a Turing machine in time $b(|\vec{w}|)$, for some $b \in \mathcal{E}^n$. As a consequence of Theorem 4.1, and similarly to what has been recalled in Section III, the following result holds.

Theorem 4.2: For $n \geq 0$: $\mathcal{L}^n_\sigma = \mathcal{G}^{n+2}$.

V. Conclusions and Future Work

We have defined a syntactical measure σ that considers how the iteration of imperative stack programs affects the complexity of the programs themselves. In particular, this measure only counts those loops in which programs with a size-increasing effect (w.r.t. the final length of the result) are iterated. Each time such a loop is built over other loops, the σ-measure is increased by 1. Other measures detect potentially harmful loops, but are not able to distinguish between size-increasing loops and the non-size-increasing one's. It is undecidable to know if a function computed by a given stack program lies in a given complexity class, but our measure represents an improvement when compared to previously defined measures. We can assign a function computed by a stack program of σ-measure n to the $n+2-th$ Grzegorczyk class, and this class is lower in the hierarchy, when compared to the class obtained via the μ-measure.

References

[1] S. Bellantoni and S. Cook, "A new recursion-theoretic characterization of the poly-time functions," Computational Complexity, no. 2, pp. 97-110, 1992.
[2] P. Clote, "Computation models and function algebra," in E. Grivor (Ed.), Handbook of Computability Theory, Elsevier, Amsterdam, 1996.
[3] A. Cobham, "The intrinsic computational difficulty of functions," in Y. Bar-Hillel (ed), Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science, pp. 24-30, North-Holland, Amsterdam, 1962.

[4] E. Covino and G. Pani, "Diagonalization and the complexity of programs," The Ninth International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking, (COMPUTATION TOOLS 2018), February 18-22, 2018, Barcelona, Spain. ISBN: 978-1-61208-394-0. ISSN: 2308-4170

[5] A. Grzegorczyk, "Some classes of recursive functions," Rozprawy Mat., Vol. IV, Warszawa, 1953.

[6] M. Hofmann, "The strength of non-size-increasing computations," Principles of Programming Languages, POPL'02, Portland, Oregon, January 16-18th, 2002.

[7] N. Jones, "Program analysis for implicit computational complexity," Third International Workshop on Implicit Computational Complexity (ICC'01), Aarhus.

[8] N. Jones, "LOGSPACE and PTIME characterized by programming languages," Theoretical Computer Science, no. 228, pp. 151-174, 1999.

[9] L. Kristiansen, "New recursion-theoretic characterizations of well known complexity classes," Fourth International Workshop on Implicit Computational Complexity (ICC'02), Copenhagen.

[10] L. Kristiansen and K.-H. Niggl, "On the computational complexity of imperative programming languages," Theoretical Computer Science, no. 318(1-2), pp. 139–161, 2004.

[11] L. Kristiansen and K.-H. Niggl, "The garland measure and computational complexity of imperative programs," Fifth International Workshop on Implicit Computational Complexity, (ICC '03), Ottawa.

[12] D. Leivant, "A generic imperative language for polynomial time," arXiv:1911.04026v2 [cs.LO], 2020.

[13] D. Leivant, "Subrecursion and lambda representation over free algebras," in S. Buss, P. Scott (Eds.), Feasible Mathematics, Perspectives in Computer Science, BirkhLauser, Boston, New York, 1990, pp. 281–291.

[14] D. Leivant, "A foundational delineation of computational feasibility," in Proc. Sixth IEEE Conf. on Logic in Computer Science (Amsterdam), IEEE Computer Society Press, Washington, DC, 1991.

[15] D. Leivant, "Stratifed functional programs and computational complexity," in Conf. Record of the 20th Annual ACM Symposium on Principles of Programming Languages, New York, 1993, pp. 325–333.

[16] D. Leivant, "Ramifed recurrence and computational complexity I: Word recurrence and poly-time," in P. Clote, J. Remmel (Eds.), Feasible Mathematics II, Perspectives in Computer Science, BirkhLauser, Basel, 1994, pp. 320–343.

[17] D. Leivant and J.-Y. Marion, "Primitive recursion in the abstract," Mathematical Structures in Computer Science, Cambridge University Press (CUP), 2020, 30 (1), pp. 33-43. 10.1017/S0960129519000112. hal-02573188.

[18] D. Leivant and J.-Y. Marion, "Ramified recurrence and computational complexity II: substitution and polyspace," in J. Tiuryn and L. Pocholsky (eds), Computer Science Logic, LNCS no. 933, pp. 486-500, 1995.

[19] A. Meyer and D. Ritchie, "The complexity of loop programs," in Proceedings of the 1967 22nd National Conference, pp. 465–469, New York, NY, USA, 1967, ACM.

[20] K.-H. Niggl, "Control structures in programs and computational complexity," Fourth Implicit Computational Complexity Workshop (ICC'02), Copenhagen.

[21] I. Oitavem, "New recursive characterization of the elementary functions and the functions computable in polynomial space," Revista Matematica de la Universidad Complutense de Madrid, no. 10.1, pp. 109-125, 1997.

[22] H. E. Rose, Subrecursion: functions and hierarchies, Oxford University Press, Oxford, 1984.

[23] H. Simmons, "The realm of primitive recursion," Arch. Math. Logic 27 (1988), pp. 177–188.

 ISBN: 978-1-68558-158-9

Contextual Categorization Enhancement through LLMs Latent-Space

Zineddine Bettouche, Anas Safi, Andreas Fischer
Deggendorf Institute of Technology
Dieter-Görlitz-Platz 1, 94469 Deggendorf
zineddine.bettouche@th-deg.de, anas.safi@stud.th-deg.de, andreas.fischer@th-deg.de

***Abstract*—Managing the semantic quality of the categorization in large textual datasets, such as Wikipedia, presents significant challenges in terms of complexity and cost. In this paper, we propose leveraging transformer models to distill semantic information from texts in the Wikipedia dataset and its associated categories into a latent space. We then explore different approaches based on these encodings to assess and enhance the semantic identity of the categories. Our graphical approach is powered by Convex Hull, while we utilize Hierarchical Navigable Small Worlds (HNSWs) for the hierarchical approach. As a solution to the information loss caused by the dimensionality reduction, we modulate the following mathematical solution: an exponential decay function driven by the Euclidean distances between the high-dimensional encodings of the textual categories. This function represents a filter built around a contextual category and retrieves items with a certain Reconsideration Probability (RP). Retrieving high-RP items serves as a tool for database administrators to improve data groupings by providing recommendations and identifying outliers within a contextual framework.**

***Index Terms*—Natural Language Processing, Contextual Categorization, Large Language Models, BERT, Convex Hull, Hierarchical Navigable Small Worlds, High-dimensional Latent Space, Dimensionality Reduction.**

I. Introduction

In the modern age of abundant textual data, effective categorization poses a significant challenge due to its increasing complexity. As data volumes grow exponentially, traditional methods struggle to handle the task adequately.

Fortunately, advancements in Natural Language Processing (NLP) provide a promising solution. NLP techniques, like word embeddings, encode semantic information, enabling automated and accurate categorization of vast datasets. Long Short-Term Memory (LSTM) networks, a type of recurrent neural network, excel at modeling sequential data and have proven effective in automating categorization tasks.

Transformer models [1], such as the BERT series [2], offer a deep understanding of contextual nuances, enhancing categorization accuracy. Innovative algorithms like Convex Hull [3] and Hierarchical Navigable Small World (HNSW) [4], powered by embeddings, can be employed to check the categorization efficiency by organizing and navigating through high-dimensional spaces.

In our previous works [5], [6], we addressed the contextual clustering of transformer encodings in an unlabeled database of scientific articles. In this study, we investigate the effectiveness of these methodologies—NLP techniques, LLMs, and geometric algorithms—in improving categorization efficiency and accuracy. Through empirical analysis, we aim to demonstrate their efficacy in managing large-scale data categorization challenges.

As for the structure of this paper, Section II offers a background on key topics such as Transformer models, and the NLP techniques employed. Section III cites the related works, setting a foundation and context for the research. Section IV lays out the methodology, detailing the approach, models, and metrics used. Section V presents the experiments, the challenges faced, trade-offs considered, and the results derived from the techniques employed. Finally Section VI concludes the study and sets up future work.

II. Background

This section introduces the background of the techniques used to develop our methodology.

A. Transformer Models: BERT

Transformer models, such as BERT, are pivotal tools in the field of natural language processing. BERT, which stands for Bidirectional Encoder Representations from Transformers, has transformed the way we handle text data. BERT models are pretrained on vast textual corpora and excel at converting text into high-dimensional vectors. They shine at capturing intricate semantic relationships between words and sentences, making them invaluable for various NLP tasks.

B. Convex Hull

Convex hulls are key concepts in computational geometry and mathematics. They represent a closed, convex shape that encloses a set of data points in multi-dimensional space. In our study, we use convex hulls to define boundaries around groups of BERT-encoded articles, creating distinct regions within the latent space. This method aids in exploring how articles are distributed within a specific category and identifying those that are positioned near the boundaries. This enables us to adjust and extend category definitions effectively.

C. Hierarchical Navigable Small Worlds

Hierarchical Navigable Small Worlds (HNSWs) serve as a data structure for approximate similarity searches in high-dimensional spaces. They offer a practical and scalable means to navigate complex, multi-dimensional data while preserving

 ISBN: 978-1-68558-158-9

proximity relationships. In our research, HNSWs are employed to efficiently organize and search BERT-encoded vectors. This facilitates the process of identifying articles that closely resemble a given category within the latent space. The hierarchical nature of HNSWs enhances our ability to adjust and expand categories based on latent similarities.

III. Related Work

In this section, we provide a concise overview of related work, highlighting key contributions that inform our study.

Moas' investigation into "Real-time prediction of Wikipedia articles' quality" [7] examines various assessment methods, automated and manual, highlighting the challenges in manual assessment due to its time-consuming nature and issues like "circular categories." The paper suggests using an Application Programming Interface (API) to address these challenges in analyzing category trees. Another study, "The Use of NLP-Based Text Representation Techniques to Support Requirement Engineering Tasks," [8] discusses levels of Natural Language Processing (NLP). It emphasizes the semantic level for understanding text meaning and supports the adoption of the Vector Space Model [9] for its simplicity in representing articles in high-dimensional spaces.

The paper "Data Mining with Python" [10] by Nielsen highlights NetworkX and DiGraphs' utility in analyzing hierarchical structures like category trees in Wikipedia. These directed graphs accurately represent parent-child relationships, essential in modeling category relationships. Studies on convex hulls, such as Preparata and Hong's computational aspects [11] and Graham's algorithm [3], provide insights into computational methods for points in two and three-dimensional spaces. Understanding convex hull derivation from simple polygons aids in comprehending data distribution in high-dimensional spaces, relevant to machine learning.

The research by Malkov and Yashunin [4] introduces Hierarchical Navigable Small World (HNSW) graphs for kNN search, implemented in this thesis for vector retrieval. The analysis revealed that articles within the same category tend to be the closest neighbors in the constructed HNSW structure.

Johnson et al. [12] proposed a language-agnostic method for categorizing Wikipedia articles, overcoming content quality and geographic variations. Leveraging article links, their approach matches language-dependent methods in performance, extending coverage across Wikipedia languages. Biswas et al. [13] introduced Cat2Type, enhancing entity typing in knowledge graphs (KGs) like DBpedia and Freebase using Wikipedia categories. By utilizing semantic information, Cat2Type surpasses existing methods on real-world datasets. Ostendorff et al. [14] addressed semantic relationship identification between documents, treating it as a pairwise classification task. Employing BERT and XLNet, experiments on Wikipedia article pairs and Wikidata properties validated BERT's effectiveness, suggesting potential for semantic document-based recommender systems.

This overview summarizes findings from various research domains, including article categorization, NLP techniques, dimensionality reduction, graph analysis, convex hulls, and nearest neighbor search, all relevant to this study. Notably, the reviewed research does not directly address the use of transformer-generated encodings in their high-dimensionality (full information), which differentiates the methodology presented in this paper. The paper introduces a novel mathematical function that utilizes these encodings, representing a distinctive contribution to the field.


```
{
 "id": "17279752",
 "text": "Hawthorne Road was a cricket and football ground in Bootle in England.
 "title": "Hawthorne Road"
},
{
 "id": "17279785",
 "text": "\"Nothing Lasts Forever\" is a single by Echo & the Bunnymen which was
 "title": "Nothing Lasts Forever (Echo &amp;amp; the Bunnymen song)"
},
{
 "id": "17279795",
 "text": "KPTY (1330 AM, \"107.3 Hank FM\") is a radio station serving the Water
 "title": "KPTY (AM)"
},
```


Figure 1. Sample of Data Objects in the Wikipedia Dataset

IV. Methodology

This section presents the Wikipedia dataset used in this work and the approaches developed to select articles for context-based reconsideration.

A. Overview of Wikipedia Dumps

The data used is from the Wikipedia dump of November 2020 on Kaggle [15] (plain text). There are 6,144,363 articles in the dataset divided into 605 JSON files. Figure 1 shows a random sample of items in these JSON files. Each item has an id, a text, and a title.

The transformer model processes articles individually, ensuring consistent and undistorted encodings regardless of the number of articles processed and sample size. Therefore, as a proof of concept and for computational feasibility, we randomly selected 10,000 articles to assess the approaches and conduct experiments.

B. Convex Hull

Our second approach involves the construction of a convex hull on the set of vectors representing the category. The vectors of the category tree are mapped onto a 2D plane using UMAP [16]. The reason behind including UMAP is that it is not feasible to construct a convex hull in the 768 dimensions Every article in the dataset undergoes encoding and mapping onto the 2D plane, enabling the determination of whether any article breaches the established convex hull boundary. Convex hulls provide an efficient way to capture the spatial relationships between category vectors when mapped onto a 2D plane. This approach leverages the geometric properties of convex hulls, enabling the visualization and definition of the boundaries of category clusters. By determining whether an article breaches the established convex hull boundary, one can promptly identify articles that do not conform to their designated categories.

 ISBN: 978-1-68558-158-9

C. HNSW

The application of Hierarchical Navigable Small World (HNSW) is used in the third approach to retrieve the nearest neighbors of articles within a category tree. The premise of this approach is that retrieved articles should already share the same category. The presence of articles that are retrieved but do not belong to the category tree should be reconsidered for category addition. We construct an HNSW using the randomly selected articles, alongside category vectors. For each category vector, we retrieve the five nearest neighbors in this HNSW. Articles that appear and are not associated with the category, are flagged for inspection. This approach leverages the structure of the data. Articles within the same category should naturally be closer to each other in the vector space. By flagging articles retrieved by HNSW that do not belong to the category tree, one can promptly identify instances of lack of categorization.

D. Filter built on High-dimensional Latent Space

The high dimensionality of the encodings can be too complex for several techniques, and it is referred to as *the curse of dimensionality* in this case. However, we intend to use these information-rich vectors to our advantage. We design a filter that takes the category articles (as encodings) and its centroid vector. This filter is applied on the articles of the sample. It takes a percentage that represents the Reconsideration Probability (RP).

An article with 100% RP must be reconsidered to be added to the concerned category. We encode the category into latent space and calculate the centroid vector. The category encodings form a cloud around this centroid vector in the latent space. Any non-category vector in the dataset that has a distance to the centroid vector equal to or less than the radius of this category cloud is assigned 100% RP.

We assume that in the sample there must at least one article that must not be in the selected category. We assign this non-category article 0.1% RP (0% RP complicates the mathematical modeling of the filter). In the case that the dataset admin cannot select such an article, we assign 0.1% RP to the article with the farthest encoding from the centroid.

As for the filter function, let d_c represent the distance of the last quarter threshold (75th percentile). This value is often used as a measure of central tendency that is less affected by outliers compared to the mean. Let d_{ea} represent the distance of the examined article. The $RP(d_c, d_{ea})$ should be 100% when d_{ea} is equal to d_c, and approach 0% the greater is d_{ea} compared to d_c. We achieve this is by using an exponential decay function with a horizontal shift. To determine the decay constant k, we need to consider the set of non-category articles and their distances. The median value $median(set(d_{ea}))$ in this set of distances should result in 50% RP. Equations 1 and 2 show the mathematical modeling of this description.

$$k = \frac{-ln(0.5)}{median(set(d_{ea})) - d_c} \quad (1)$$

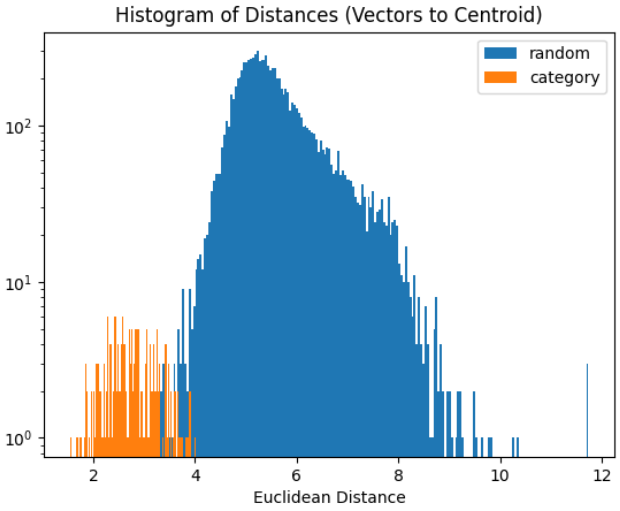

Figure 2. Euclidean Distances between Centroid and Space Vectors

$$RP(d_c, d_{ea}) = 100 * e^{-k*(d_{ea}-d_c)} \quad (2)$$

V. Experiments

This section discusses the experiments done in this paper.

A. Setting up the Vector Space: Encodings of the Category and the Sample Articles

To set the ground for the experiments, the sample of articles and the category articles are all encoded into the vector space. The centroid vector of the category is calculated by averaging its vectors. We calculate then the euclidean distances between this centroid vector and every other vector in this space. Figure 2 shows the results of these calculations. We observe that the distances of the sample articles fall mostly in the interval of distances: $[3.5, 10.5]$ while the category articles have distances to their centroid not exceeding 4, showing an overlap in $[3.5, 4]$. The upcoming experiments shed more light on how to retrieve the closest articles to the centroid (other than the category vectors), and we highlight how these retrieved articles fall distance-wise.

B. Convex Hull: Geometric Boundaries

In this experiment we construct the convex hull with the encodings of the category (Figure 3). The convex hull is placed on the map of the 10,000 articles and the articles that breach it are recorded (Figure 4). The distance-to-centroid of each article breaching the convex hull is recorded and plotted in Figure 5.

The convex hull successfully identified 10 out of the 55 articles with distances smaller than 4 and 33 with distances smaller than 5. This outcome showcases a downside to consider, as the convex hull also selected articles positioned farther away from the centroid than the articles it ignored. This is a form of "blindness" in the method, warranting further exploration and refinement.

 ISBN: 978-1-68558-158-9

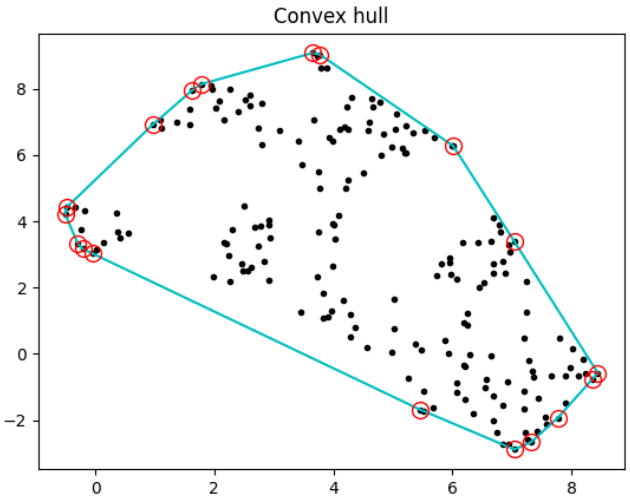

Figure 3. Convex Hull of the Category

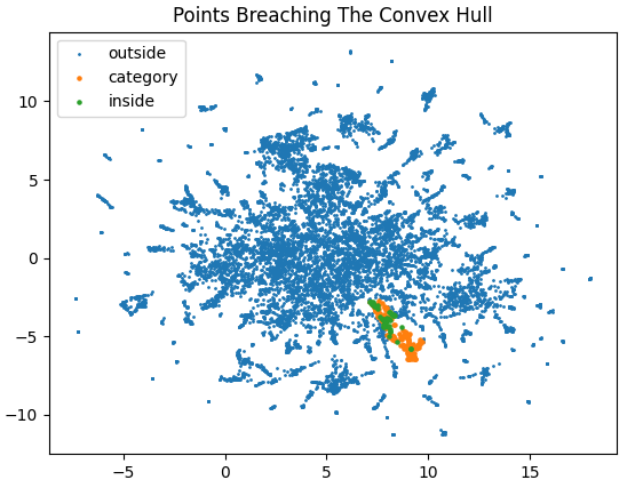

Figure 4. Map of Articles Breaching the Convex Hull

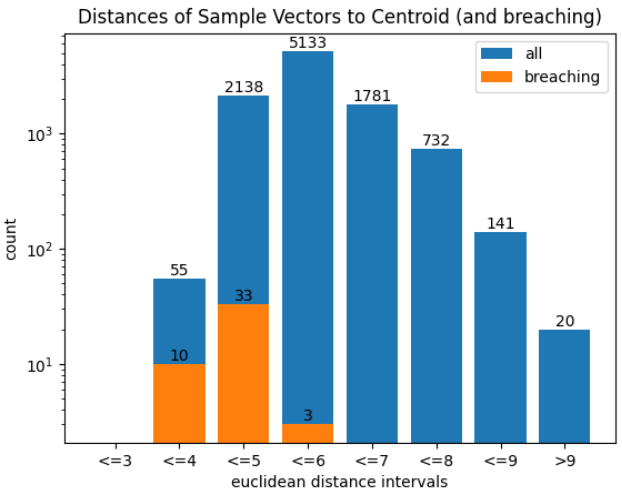

Figure 5. Histogram of Distances of Convex-Hull-Breaching Articles to Category Centroid

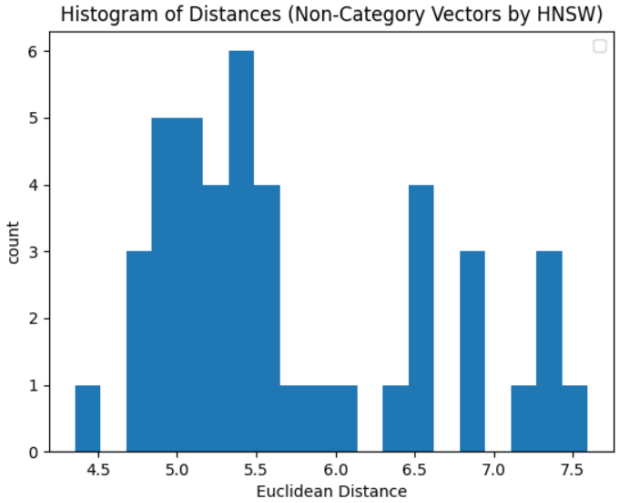

Figure 6. Histogram of Distances of HNSW-Retrieved Articles (non-category) to Category Centroid

C. The Contextual Category in an HNS-World of Articles

An HNSW is built on the setup vectors (sample articles, category articles, and the centroid vector). We retrieve iteratively all of the category vectors from the position of the centroid vector in the HNSW. In this process, we consider the category vectors as a fishnet to retrieve within any other non-category vector that is closer to the centroid vector than the farthest category-vector.

We calculated the distances of the retrieved non-category vectors to the category centroid (Figure 6). We observe that the built HNSW jumped over articles closer to the centroid (only one with $d < 4.5$) and retrieved farther ones (six with $d = 5.4$). This indicates that HNSW follows a pattern similar to the convex hull technique. This similarity arises from the implicit mapping performed by HNSW, resulting in a mapping structure mirroring the loss of information already seen in the convex hull technique.

D. High-Dimensional Latent-Space Filter

As described in IV-D, the 75th percentile distance from the centroid vector to articles of the category d_c is 3.151 (Figure 7). The median of the distances in the non-category vectors $median(set(d_{ea}))$ is 5.447. By numerical application, Equations 1 and 2 are as follows:

$$k = \frac{-ln(0.5)}{median(set(d_{ea})) - d_c} = 0.302$$

$$RP(d_c = 3.151, d_{ea}) = 100 * e^{-0.302*(d_{ea}-3.151)}$$

We calculate the RP values of every articles in the sample (Figure 8). The distances of these articles d_{ea} are in the range of $[3.143, 11.737]$:

- For the minimum value: $RP(3.151, 3.143) = 100.241\%$ (saturated to 100%)

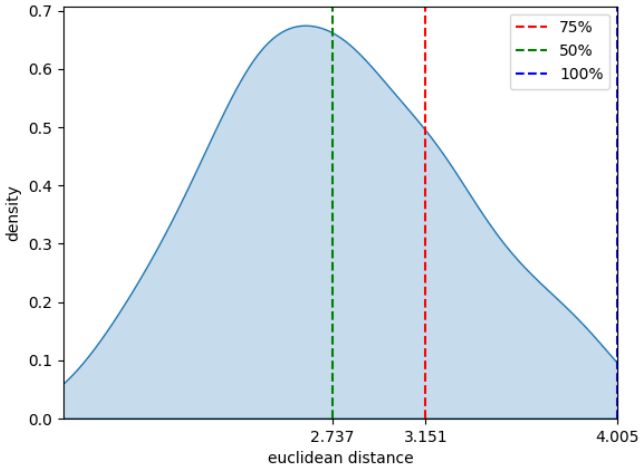

Figure 7. Distances of Category Articles to Category Centroid

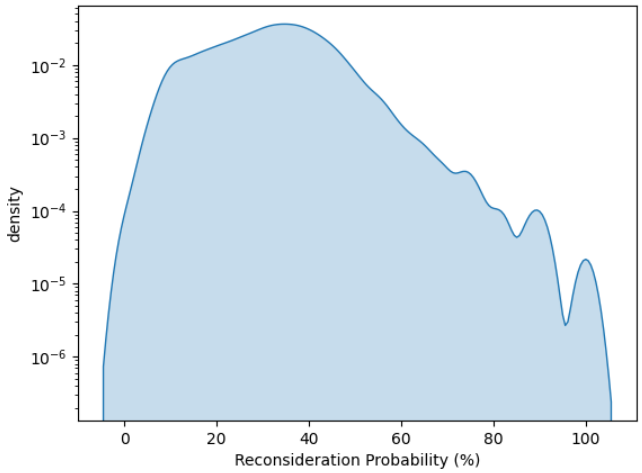

Figure 8. RP of Sample Articles to Category Centroid

- For the median value: $RP(3.151, 5.447) = 49.988\%$
- For the maximum value: $RP(3.151, 11.737) = 7.479\%$

We take the articles with $RP > 75\%$, extract their keywords and plot their wordcloud (Figure 9). The filter retrieved 20 articles with this threshold. Table I shows the breakdown of closest 5 retrieved items. The category "Serbian Films" is similar in context to art topics from the Balkans. This explains the presence of articles about writings from the ex-Yugoslavian countries. To test the variations in centroid vector, we sample the data into 100 samples. The mean distance between the centroid vector and the 80% sample centroid vector is 0.102 with a standard variation of 0.026. This shows an initial stability in the category.

E. Bonus: Hierarchical Vectors Effect on Cluster Cohesion

The presence of hierarchical structures in the form of categories and sub-categories raises the question regarding their utility. One hypothesis worth exploring is whether the incorporation of hierarchical vectors into clusters can reinforce their internal coherence, rendering them more distinct

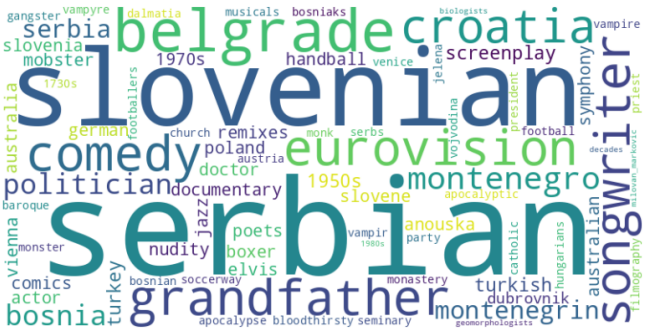

Figure 9. Wordcloud of Articles with $RP > 75\%$

Table I
CLOSEST 5 FILTER-RETRIEVED ARTICLES (CATEGORY: *Serbian Films*)

RP (%)	**Title**	**Keywords**
100.00	Croatia in Eurovision 2006	eurovision serbia montenegro
91.216	Đani Pervan	bosnian musician songwriter
90.993	Vranjic	venice church
90.499	Vinci Vogue Anžlovar	vampyre blog slovenian
89.713	Jovan Cvijić	serbian ethnological historic

from one another. To quantify this distinction, the Silhouette score [17] serves as a reliable metric. To initiate the clustering process, two distinct clusters were created based on articles encoded with BERT, resulting in an initial Silhouette score of 0.23. Subsequently, hierarchical vectors were calculated to represent sub-categories (Figure 10). Each sub-category's vector was computed as the average vector of its constituent encoded articles (further elucidated in the subsequent section). These hierarchical vectors were integrated into the clustering process, and the clustering was re-executed. Notably, the new Silhouette score was improved by 13% (0.26). This shows the impact of incorporating hierarchical vectors, leading to denser clusters.

VI. CONCLUSION AND FUTURE WORK

In conclusion, this paper has addressed the goal of improving the efficiency of contextual categorization within hierarchical structures. This work employed the Wikipedia dumps and its categories, along with BERT models as the latent-

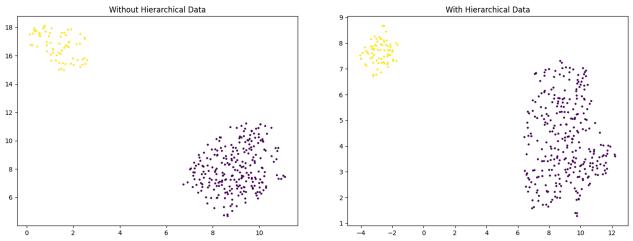

Figure 10. 2D Mappings of two different categories (left: without hierarchical vectors, right: with them).

representation technique. The exploration has focused on the integration of hierarchical vectors and advanced clustering techniques.

The experiments showed the practicality of calculating centroid vectors for category trees. The centroid vector, obtained by averaging all the vectors within a category tree, served as a key item for assessing the proximity of articles to the tree. The relationship between an article's vector and the centroid vector provided a quantitative basis for evaluating the relevance of an article to a specific category tree. This approach allowed for a reconsideration of category labels based on the calculated distances. Alternative techniques such as convex hulls and HNSWs, although explored, exhibited distortions in the results due to the inherent mapping processes involved. To overcome the loss of information and take advantage of the high-dimensionality of the embeddings, we modulated a filter using the exponential decay function that indicates the Reconsideration Probability. The transformer model processes articles individually, ensuring consistent and undistorted encodings regardless of the number of articles processed and sample size. Therefore, the scalability of the exponential decay function (our modulated filter), leveraging transformer embeddings, due to its mathematical nature, offers efficiency gains compared to the scalability challenges typically associated with convex hull and HNSW algorithms. Finally, utilizing hierarchical vectors for subcategories proved valuable, enhancing the representation of the category tree in the latent space. This was evident in the increased Silhouette score, indicating a clearer categorization structure. This utilization is not limited to our case (Wikipedia data), but extends to any form of dataset hosting such hierarchies.

Future research should refine hierarchical vector integration, develop specialized clustering algorithms for complex structures, scale experiments to larger datasets, explore new content categorization tools, assess their impact on platforms like Wikipedia and other databases more scientifically oriented, and enhance categorization systems' precision and utility from a contextual perspective.

ACKNOWLEDGEMENT

This paper has received funding from the state of Bavaria in the context of project SEMIARID, funding no. DIK-2104-0067//DIK0299/01

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Bidirectional encoder representations from transformers," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019.

[3] R. L. Graham and F. F. Yao, "Finding the convex hull of a simple polygon," in *Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science*, 1983, pp. 324–331.

[4] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," in *Proceedings of the 34th International Conference on Machine Learning*, 2018, pp. 824–836.

[5] Z. Bettouche and A. Fischer, "Mapping researcher activity based on publication data by means of transformers," in *Proceedings of the Interdisciplinary Conference on Mechanics, Computers and Electrics (ICMECE 2022)*, 2022.

[6] ——, "Topical clustering of unlabeled transformer-encoded researcher activity," *Bavarian Journal of Applied Sciences*, no. 6, pp. 504–525, 2023.

[7] P. M. B. B. L. Moas, "Real-time prediction of wikipedia articles' quality," 2022.

[8] R. Sonbol, G. Rebdawi, and N. Ghneim, "The use of nlp-based text representation techniques to support requirement engineering tasks: A systematic mapping review," 2022.

[9] G. Salton, "Automatic text processing-addison-wesley longman publishing co., inc." 2022.

[10] F. A. Nielsen, "Data mining with python (working draft)," 2017.

[11] F. P. Preparata and S. J. Hong, "Convex hulls of finite sets of points in two and three dimensions," in *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, 1977, pp. 87–93.

[12] I. Johnson, M. Gerlach, and D. Sáez-Trumper, "Language-agnostic topic classification for wikipedia," in *Companion Proceedings of the Web Conference 2021*, ser. WWW '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 594–601. [Online]. Available: https://doi.org/10.1145/3442442.3452347

[13] R. Biswas, R. Sofronova, H. Sack, and M. Alam, "Cat2type: Wikipedia category embeddings for entity typing in knowledge graphs," in *Proceedings of the 11th Knowledge Capture Conference*, ser. K-CAP '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 81–88. [Online]. Available: https://doi.org/10.1145/3460210.3493575

[14] M. Ostendorff, T. Ruas, M. Schubotz, G. Rehm, and B. Gipp, "Pairwise multi-class document classification for semantic relations between wikipedia articles," in *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries in 2020*, ser. JCDL '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 127–136. [Online]. Available: https://doi.org/10.1145/3383583.3398525

[15] DavidShapiro, "Plain text wikipedia dataset 202011," https://www.kaggle.com/datasets/ltcmdrdata/plain-text-wikipedia-202011, 2020, accessed on November 2, 2023.

[16] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection," *arXiv preprint arXiv:1802.03426*, 2018.

[17] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987.

 ISBN: 978-1-68558-158-9