COMPUTATION TOOLS 2018

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

ISBN: 978-1-61208-613-2

February 18 - 22, 2018

Barcelona, Spain

COMPUTATION TOOLS 2018 Editors

Glenn Luecke, Iowa State University, USA

Jaime Lloret Mauri, Polytechnic University of Valencia, Spain

COMPUTATION TOOLS 2018

Forward

The Ninth International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking (COMPUTATION TOOLS 2018), held between February 18 - 22, 2018 - Barcelona, Spain, 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.

The conference had the following tracks:

- Advanced computation techniques
- Tools for distributed computation

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 2018 technical program committee, as well as the numerous reviewers. The creation of such a high 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 2018. 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 2018 organizing committee for their help in handling the logistics and for their work that made this professional meeting a success.

We hope COMPUTATION TOOLS 2018 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 Barcelona provided a pleasant environment during the conference and everyone saved some time for exploring this beautiful city.

COMPUTATION TOOLS 2018 Chairs

COMPUTATION TOOLS 2018 Steering Committee

Ricardo Rocha, University of Porto, Portugal
Cristian Stanciu, University Politehnica of Bucharest, Romania
Ekaterina Komendantskaya, Heriot-Watt University, UK
Ralph Müller-Pfefferkorn, Technische Universität Dresden, Germany
Laura Carnevali, University of Florence, Italy

COMPUTATIONAL TOOLS 2018 Industry/Research Advisory Committee

Miroslav Velev, Aries Design Automation, USA
Cornel Klein, Siemens AG, Germany
Laura Nenzi, TU Wien, Austria
Cecilia Esti Nugraheni, Parahyangan Catholic University, Indonesia
Azahara Camacho, Carbures Defense, Spain
Keiko Nakata, SAP SE - Potsdam, Germany

COMPUTATION TOOLS 2018

Committee

COMPUTATION TOOLS 2018 Steering Committee

Ricardo Rocha, University of Porto, Portugal
Cristian Stanciu, University Politehnica of Bucharest, Romania
Ekaterina Komendantskaya, Heriot-Watt University, UK
Ralph Müller-Pfefferkorn, Technische Universität Dresden, Germany
Laura Carnevali, University of Florence, Italy

COMPUTATIONAL TOOLS 2018 Industry/Research Advisory Committee

Miroslav Velev, Aries Design Automation, USA
Cornel Klein, Siemens AG, Germany
Laura Nenzi, TU Wien, Austria
Cecilia Esti Nugraheni, Parahyangan Catholic University, Indonesia
Azahara Camacho, Carbures Defense, Spain
Keiko Nakata, SAP SE - Potsdam, Germany

COMPUTATION TOOLS 2018 Technical Program Committee

Davide Arcelli, University of L'Aquila, Italy
Lorenzo Bettini, DISIA - Università di Firenze, Italy
Ateet Bhalla, Independent Consultant, India
Narhimene Boustia, University Saad Dahlab, Blida 1, Algeria
Azahara Camacho, Carbures Defense, Spain
Laura Carnevali, University of Florence, Italy
Emanuele Covino, Università degli Studi di Bari Aldo Moro, Italy
Marc Denecker, KU Leuven, Belgium
David Doukhan, Institut national de l'audiovisuel (Ina), France
António Dourado, University of Coimbra, Portugal
Andreas Fischer, Technische Hochschule Deggendorf, Germany
Tommaso Flaminio, DiSTA - University of Insubria, Italy
Khalil Ghorbal, INRIA, Rennes, France
George A. Gravvanis, Democritus University of Thrace, Greece
Fikret Gurgen, Bogazici University - Istanbul, Turkey
Hani Hamdan, Université de Paris-Saclay, France
Cornel Klein, Siemens AG, Germany
Ekaterina Komendantskaya, Heriot-Watt University, UK
Roderick Melnik, Wilfrid Laurier University, Canada
Ralph Müller-Pfefferkorn, Technische Universität Dresden, Germany
Keiko Nakata, SAP SE, Germany

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

Diagonalization and the Complexity of Programs

Emanuele Covino and Giovanni Pani
Dipartimento di Informatica
Università di Bari, Italy
Email: emanuele.covino@uniba.it, giovanni.pani@uniba.it

***Abstract*—Starting from the definitions of *predicative recursion* and *constructive diagonalization*, we recall our specialized programming language that provides a resource-free characterization of register machines computing their output within polynomial time $O(n^k)$, and exponential time $O(n^{n^k})$, for each finite k. We discuss the possibility of extending this characterization to a transfinite hierarchy of programs that captures the Grzegorczyk hierarchy of functions at elementary level. This is done by means of predicative operators, contrasting to previous results. We discuss the feasibility and the complexity of our diagonalization operator.**

***Index Terms*—Hierarchies of complexity classes; Predicative recursion; Constructive diagonalization.**

I. Introduction and position of the problem

In [1], the Grzegorczyk's classes of functions $\mathcal{E}^n$ ($n = 0, 1, \ldots$), have been introduced, with the property that $\bigcup_{n<\omega} \mathcal{E}^n$ is the class of primitive recursive functions. In order to define these classes, the hierarchy functions E_n have to be introduced. They are essentially repeated iterations of the successor function, i.e., $E_0(x, y) = x + y$, $E_1(x) = x^2 + 2$, $E_{n+2}(0) = 2$, $E_{n+2}(x + 1) = E_{n+1}(E_{n+2}(x))$. Grzegorczyk's classes can be defined as follows. $\mathcal{E}^0$ is the class whose initial functions are the zero, successor and the projection functions, and is closed under composition and limited recursion. $\mathcal{E}^{n+1}$ is defined similarly, except that the function E_n is added to the list of the initial functions. $\mathcal{E}^n$ is called the n-th Grzegorczyk class. Other sequences of hierarchy functions have been used in literature, for instance the Ackermann function. The reader can find properties and theorems in [2]; we recall that the class $\mathcal{E}^3$ is the class of the elementary functions $\mathcal{E}'$ (that is, the class of functions containing the successor, projections, zero, addition, multiplication, subtraction functions, and closed under composition and bounded sum and product.)

Harmonizations of significant complexity classes with the Grzegorczyk classes have been obtained by Leivant [3], Niggl [4], and Bellantoni and Niggl [5]. In these papers, the class of polynomial-time computable functions is characterized by means of different definitions of *predicative recursion* [6] or *ramified recurrence* [7], and starting from a set of initial functions. Note that a predicative definition of a recursive function is based on the idea that functions have two kind of variables: those whose values are known entirely (and which can be recursed upon, for instance), and those whose values are still being computed (and are accessible in a more restricted way, on the least significant digits, for instance); these two types of variables are called safe and normal in [6] (dormant and normal in [8]); roughly speaking, normal variables are used only for recursive calls, while safe variables are used only for substitution. This allows to discard explicitly bounded schemes (like the limited recursion) to characterize classes of functions.

In [5], the classes $\mathcal{E}^n$ are captured by closure under composition of functions, and by counting the number of infringements to the predicative principle made into the recursive definitions. This is an alternative approach to look at ramification: rather than controlling the type of the variables, and how they are used in the definition of a recursive function, first a definition of a function is given, and then one examines it in order to see, or to count, how many levels of impredicative definitions are used. Even if this approach represents a detailed analysis of the effects of nesting recursive definitions, we believe that counting the number of violations to the predicative principle is as impredicative as using limited recursion, or as adding a hierarchy function to the list of initial functions.

In a previous paper [9], we introduced our version of predicative recursion, together with a *constructive diagonalization* operator; they allow us to define a hierarchy of programs $\mathcal{T}_k$ ($k = 0, 1, \ldots$), such that each program defined in $\mathcal{T}_k$ is computable by a register machine within time bounded by a polynomial n^k; and to extend this hierarchy up to the programs computable within exponential-time bound. In this contribution we claim that our approach can be extended further, and that our hierarchy reaches the level 3 of the Grzegorczyk hierarchy. We address questions raised about the complexity and feasibility of the diagonalization, and we compare it to a recent different approach [10].

In Section II, we recall the definition of our programming language, and the results holding on the finite levels of our hierarchy of programs. In Section III, we introduce the definition of diagonalization, we discuss its feasibility, and we recall the result on programs with exponential-time complexity. In Section IV, we extend the hierarchy of programs up to the elementary level of the Grzegorczyk hierarchy. In Section V, we compare our approach to Marion's [10]. Conclusions and further work are in Section VI.

 ISBN: 978-1-61208-613-2

II. BASIC INSTRUCTIONS, COMPOSITION OF PROGRAMS AND RECURSION

In this section, we recall the basic instructions of our programming language, together with the definition schemes of composition and recursion over programs we introduced in [9]. We then recall the definition of our finite hierarchy of programs, which captures the polynomial-time computable functions.

The language is built over lists of binary words, with the symbol © acting as a separator between each word. $\mathbf{B}$ denotes the alphabet $\{0,1\}$, and $a, b, a_1, \ldots$ denote elements of $\mathbf{B}$; $U, V, \ldots, Y$ denote words over $\mathbf{B}$. $r, s, \ldots$ stand for lists in the form $Y_1©Y_2©\ldots©Y_n$. ϵ is the empty word. The *i-th component* $(s)_i$ of a list $s = Y_1©Y_2©\ldots©Y_n$ is Y_i. $|s|$ is the length of the list s, that is the overall number of symbols occurring in s.

We write x, y, z for the variables used in a program, and we write u for one among x, y, z. Programs are denoted with letters f, g, h, and we write $f(x, y, z)$ for the application of the program f to variables x, y, z, where some among them may be absent. In what follows, the *length* $lh(f)$ of a program f is the number of basic instructions and defining schemes occurring in its definition.

The basic instructions allow us to manipulate lists of words, adding digits to (or erasing parts of) each component; the so-called simple schemes allow us to change the name of some among the variables and to select between programs, according to the value of the variables. The *basic instructions* are:

1) the *identity* $\mathrm{I}(u)$ that returns the value s assigned to u;
2) the *constructors* $\mathrm{C}_i^a(s)$ that add the digit a at the right of the last digit of $(s)_i$, with $a = 0, 1$ and $i \geq 1$;
3) the *destructors* $\mathrm{D}_i(s)$ that erase the rightmost digit of $(s)_i$, with $i \geq 1$.

Constructors $\mathrm{C}_i^a(s)$ and destructors $\mathrm{D}_i(s)$ leave the input s unchanged if it has less than i components. For instance, for $s = 01©11©©00$, we have that $|s| = 9$, and $(s)_2 = 11$; we also have $\mathrm{C}_1^1(01©11) = 011©11$, $\mathrm{D}_2(0©0©) = 0©©$, and $\mathrm{D}_2(0©©) = 0©©$.

Given the programs g and h, f is defined by *simple schemes* if it is obtained by:

1) *renaming* of x as y in g, that is, f is the result of the substitution of the value of y to all occurrences of x into g. Notation: $f = \mathrm{RNM}_{x/y}(g)$;
2) *renaming* of z as y in g, that is, f is the result of the substitution of the value of y to all occurrences of z into g. Notation: $f = \mathrm{RNM}_{z/y}(g)$;
3) *selection* in g and h, when for all s, t, r we have

$$f(s,t,r) = \begin{cases} g(s,t,r) & \text{if the rightmost digit} \\ & \text{of } (s)_i \text{ is } b \\ h(s,t,r) & \text{otherwise,} \end{cases}$$

with $i \geq 1$ and $b = 0, 1$. Notation: $f = \mathrm{SEL}_i^b(g, h)$.

Simple schemes are denoted with SIMPLE. For instance, if f is defined by $\mathrm{RNM}_{x/y}(g)$ we have that $f(t, r) = g(t, t, r)$. Similarly, f defined by $\mathrm{RNM}_{z/y}(g)$ implies that $f(s, t) = g(s, t, t)$. For $s = 00©1010$, and $f = \mathrm{SEL}_2^0(g, h)$, we have that $f(s, t, r) = g(s, t, r)$, since the rightmost digit of $(s)_2$ is 0.

Given the programs g and h, the program f is defined by *safe composition* of h and g in the variable u if it is obtained by the substitution of h to u in g, if $u = x$ or $u = y$; the variable x must be absent in h, if $u = z$. Notation: $f = \mathrm{SCMP}_u(h, g)$. The rationale behind this definition will be clear as soon as we will define the safe recursion scheme.

A *modifier* is obtained by the safe composition of a sequence of constructors and a sequence of destructors, and the class $\mathcal{T}_0$ is defined by closure of modifiers under selection and safe composition. Notation: $\mathcal{T}_0$=($modifier$; SCMP, SEL). All programs in $\mathcal{T}_0$ modify their inputs according to the result of some test performed over a fixed number of digits.

Given the programs $g(x, y)$ and $h(x, y, z)$, the program $f(x, y, z)$ is defined by *safe recursion* in the *basis* g and in the *step* h if for all s, t, r we have

$$\begin{cases} f(s,t,a) &= g(s,t) \\ f(s,t,ra) &= h(f(s,t,r),t,ra), \end{cases}$$

with $a \in \mathbf{B}$. Notation: $f = \mathrm{SREC}(g, h)$.
In particular, $f(x, z)$ is defined by *iteration* of $h(x)$ if for all s, r we have

$$\begin{cases} f(s,a) &= s \\ f(s,ra) &= h(f(s,r)). \end{cases}$$

with $a \in \mathbf{B}$. Notation: $f = \mathrm{ITER}(h)$. We write $h^{|r|}(s)$ for $\mathrm{ITER}(h)(s, r)$ (i.e., the $|r|$-th iteration of h on s).

We recall that x, y and z are the auxiliary variable, the parameter, and the principal variable of a program obtained by means of the previous recursion scheme, respectively. Note also that, according to the previous definitions, the renaming of z as x is not allowed, and if the step program of a recursion is defined itself by safe composition of programs p and q, no variable x (i.e., no potential recursive calls) can occur in the function p, when p is substituted into the principal variable z of q. These two restrictions imply that the step program of a recursive definition never assigns the recursive call to the principal variable. This is the key to the polynomial-time complexity bound intrinsic to our programs, and fulfills the predicative criteria.

Given the previous basic instructions and definition schemes, we are able to define the hierarchy of classes of programs $\mathcal{T}_k$, with $k < \omega$, as follows:

1) $\mathrm{ITER}(\mathcal{T}_0)$ denotes the class of programs obtained by one application of iteration to programs in $\mathcal{T}_0$;
2) $\mathcal{T}_1$ is the class of programs obtained by closure under safe composition and simple schemes of programs in $\mathcal{T}_0$ and programs in $\mathrm{ITER}(\mathcal{T}_0)$;
   Notation: $\mathcal{T}_1$=($\mathcal{T}_0$, $\mathrm{ITER}(\mathcal{T}_0)$; SCMP, SIMPLE);

3) $\mathcal{T}_{k+1}$ is the class of programs obtained by closure under safe composition and simple schemes of programs in $\mathcal{T}_k$ and programs in SREC($\mathcal{T}_k$), with $k \geq 1$;
Notation: $\mathcal{T}_{k+1}$=($\mathcal{T}_k$, SREC($\mathcal{T}_k$); SCMP, SIMPLE).

In [11] and [9] we proved the following two theorems using as a model of computation the register machines introduced by Leivant [7] . We have that

1) each program $f(s,t,r)$ defined in $\mathcal{T}_k$ can be computed by a register machine within time bounded by the polynomial $|s| + lh(f)(|t| + |r|)^k$, with $k \geq 1$;
2) a register machine which computes its output within time $O(n^k)$ can be simulated by a program f in $\mathcal{T}_k$, with $k \geq 1$.

The previous result allowed us to prove the following

Theorem 2.1: A program f belongs to $\mathcal{T}_k$ if and only if f is computable by a register machine within time $O(n^k)$, with $k \geq 1$.

We recall that register machines are polytime reducible to Turing machines; thus, the sequence of classes $\mathcal{T}_k$ captures PTIMEF (see [6] and [7] for similar characterizations of this complexity class).

III. DIAGONALIZATION AND EXPONENTIAL-TIME COMPUTABLE PROGRAMS

We recall the definition of structured ordinals and of hierarchies of slow growing functions, as reported in [12]. Then, we give the definition of *diagonalization* at a given limit ordinal λ, based on the sequence of classes $\mathcal{T}_{\lambda_1}, \ldots, \mathcal{T}_{\lambda_n}, \ldots$ associated with the fundamental sequence of λ. A similar operator can be found in [13], and we will discuss later the relation between our diagonalization and its analogue in [10]. Using safe recursion and diagonalization, we are able to define a transfinite hierarchy of programs characterizing the classes of register machines computing their output within time between $O(n^k)$ and $O(n^{n^k})$ (with $k \geq 1$ and n the length of the input), that is, the computations with time complexity between polynomial- and exponential-time.

Following [12], we denote limit ordinals with greek small letters α, β, λ, ..., and we denote with λ_i the i-th element of the fundamental sequence assigned to λ. For example, ω is the limit ordinal of the fundamental sequence $1, 2, \ldots$; and ω^2 is the limit ordinal of the fundamental sequence $\omega, \omega 2, \omega 3, \ldots$, with $(\omega^2)_k = \omega k$.

The *slow-growing functions* $G_\alpha : \mathbb{N} \to \mathbb{N}$ are defined by the recursion

$$\begin{cases} G_0(n) & = & 0 \\ G_{\alpha+1}(n) & = & G_\alpha(n) + 1 \\ G_\lambda(n) & = & G_{\lambda_n}(n). \end{cases}$$

We slightly change the previous definition, and we define the *slow-growing functions* $B_\alpha : \mathbb{N} \to \mathbb{N}$ by the recursion

$$\begin{cases} B_0(n) & = & 1 \\ B_{\alpha+1}(n) & = & nB_\alpha(n) \\ B_\lambda(n) & = & B_{\lambda_n}(n). \end{cases}$$

Note that $B_k(n) = n^k$, $B_\omega(n) = n^n$, $B_{\omega+k}(n) = n^{n+k}$, $B_{\omega k}(n) = n^{n \cdot k}$, $B_{\omega^k}(n) = n^{n^k}$, and $B_{\omega^\omega}(n) = n^{n^n}$; moreover, we have that $B_{\alpha+\beta}(n) = B_\alpha(n) \cdot B_\beta(n)$, and that $G_{\omega^\alpha}(n) = n^{G_\alpha(n)} = B_\alpha(n)$.

The finite hierarchy $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k, \ldots$, captures the register machines that compute their output with time in $O(1), O(n), O(n^2), \ldots, O(n^k), \ldots$, respectively. Jumping out of the hierarchy requires something more than safe recursion. We already discussed in the Introduction the approach presented in [5], that is, to define a ranking function that counts the number of nested recursions infringing the predicative definition of a program; a class of time-bounded register machines can be associated to each level of the ranking. On the other hand, given a limit ordinal λ, we proposed in [9] a new operator that *diagonalizes* at level λ over the classes $\mathcal{T}_{\lambda_i}$, that is, that selects and iterates programs in a previously defined class $\mathcal{T}_{\lambda_i}$ according to the length of the input. There is no circularity in a program defined by diagonalization, and we believe that this program isn't less predicative than a program defined by safe recursion. For instance, at level ω, we select (and iterate i times) programs in the classes $\mathcal{T}_i$, where i is the length of the input; thus, the first level of diagonalization captures the class of all register machines whose computation is bounded by a polynomial. By extending this approach to the next levels of structured ordinals, we were able to reach the machines computing their output within exponential time n^{n^k}.

Given a limit ordinal λ with the fundamental sequence $\lambda_0, \ldots, \lambda_k, \ldots$, and given an enumerator program q such that $q(\lambda_i) = f_{\lambda_i}$, for each i, the program $f(x,y)$ is defined by *diagonalization* at λ if for all s, t

$$f(s,t) = \text{ITER}^{|t|}(q(\lambda_{|t|}))(s,t)$$

where

$$\begin{cases} \text{ITER}^1(p)(s,t) & = & \text{ITER}(p)(s,t) \\ \text{ITER}^{k+1}(p)(s,t) & = & \text{ITER}(\text{ITER}^k(p))(s,t). \end{cases}$$

and f_{λ_i} belongs to a previously defined class $\mathcal{C}_{\lambda_i}$, for each i. Notation: f =DIAG(λ). Note that the previous definition requires that $f_{\lambda_i} \in \mathcal{C}_{\lambda_i}$, but there aren't other requirements on how the $\mathcal{C}$'s classes are built. In what follows, we introduce our transfinite hierarchy of programs, with an important restriction on the definition of the $\mathcal{C}$'s.

Given $\lambda < \omega^\omega$, $\mathcal{T}_\lambda$ is the class of programs obtained by

1) closure under safe composition and simple schemes of programs in $\mathcal{T}_\alpha$ and programs in SREC($\mathcal{T}_\alpha$), if $\lambda = \alpha+1$; Notation: $\mathcal{T}_{\alpha+1}$=($\mathcal{T}_\alpha$, SREC($\mathcal{T}_\alpha$); SCMP, SIMPLE).
2) closure under simple schemes of programs obtained by one application of diagonalization at λ, if λ is a limit ordinal, with $f_{\lambda_i} \in \mathcal{T}_{\lambda_i}$, for each λ_i in the fundamental sequence of λ. Notation: $\mathcal{T}_\lambda$=(DIAG(λ); SIMPLE);

In [9] we proved that

1) each program $f(s,t,r)$ defined in $\mathcal{T}_\lambda$ ($\lambda < \omega^\omega$) can be computed by a register machine within time $B_\lambda(n)$;

2) a register machine which computes its output within time $O(B_\lambda(n))$ can be simulated by a program f in $\mathcal{T}_\lambda$.

We then have that

Theorem 3.1: A program f belongs to $\mathcal{T}_\alpha$ if and only if f is computable by a register machine within time $O(B_\alpha(n))$, with $\alpha < \omega^\omega$.

Two intertwined questions (not addressed in [9]) could be raised about the enumerator $q(\lambda_i)$. First, to which class does the enumerator belongs? And what are its results? For complexity reasons, it appears clear that the enumerator should be defined into the same hierarchy of classes that we are using to diagonalize; in particular, it can be defined into the first class $\mathcal{T}_{\lambda_1}$ of every sequence $\mathcal{T}_{\lambda_1}, \ldots, \mathcal{T}_{\lambda_n}, \ldots$, because it only has to write sequences of SREC's and DIAG's according to the definition on the ordinal λ_i, in order to write down the definition of $f_{\lambda_i} \in \mathcal{T}_{\lambda_i}$. This leads us to the second question: the results of any program in our language are lists of binary words, and they aren't other programs. This means that the enumerator must return the code of a program in $\mathcal{T}_{\lambda_i}$, and not the program itself. Defining a code for every element of our language is straightforward, but we must underline that every time we diagonalize over a sequence of classes, we should see the f_{λ_i}'s as codes of programs; this implies that an interpreter is concealed in the definition of diagonalization.

IV. Extending the hierarchy to the elementary functions

In [9], the classes $\mathcal{T}_\lambda$ have been defined between level 1 and ω^ω, reaching the programs computable within exponential time. This hierarchy can be extended up to the ordinal ϵ_0, with $\epsilon_0 = \omega^{\omega^{\omega^{\cdots}}} = sup\{\omega, \omega^\omega, \omega^{\omega^\omega}, \ldots\}$; in particular, $\mathcal{T}_{\epsilon_0}$ is the class of programs computable within time $O(B_{\epsilon_0}(n)) = O(n^{n^{n^{\cdots}}})$. Given that a function $f(n)$ is elementary if and only if it is computable in time bounded by $n^{n^{n^{\cdots}}}$ (see [14]), we have that $\mathcal{T}_{\epsilon_0}$ characterizes the class of the elementary functions. The proof of this result is an extension of the proof introduced in [9]; given $\lambda < \epsilon_0$, we can prove that

1) each program $f(s,t,r)$ in $\mathcal{T}_\lambda$ can be computed by a register machine within time in $O(B_\lambda(n))$;
2) every register machine computing its output within time $O(B_\lambda(n))$ can be simulated by a program f in $\mathcal{T}_\lambda$.

Lemma (1) is proved by structural induction on the ordinal λ, that can be a finite number, an ordinal $\beta + 1$, or a limit ordinal: in each case we build the register machine that computes the program f at level λ using the machines provided by the inductive hypothesis, and we compute the overall time consumption, showing that it respects the bound $B_\lambda(n)$. Lemma (2) is proved for each time-bounded register machines showing that, given a program $nxt_M \in \mathcal{T}_0$ that simulates the transition between the machine's configurations, a program in the appropriate class $\mathcal{T}_\lambda$ can be built as the iteration of nxt_M, in order to simulate the overall computation performed by the register machine itself. By (1) and (2) we have that

Theorem 4.1: A program f belongs to $\mathcal{T}_\alpha$ if and only if f is computable by a register machine within time $O(B_\alpha(n))$, with $\alpha < \epsilon_0$.

Our operators of predicative recursion and constructive diagonalization can be used to provide a fine hierarchy of classes between PTIME and $\mathcal{E}^3$. As far as we know, this is the only characterization of these classes built "from below", by means of constructive operators. Other characterizations, like Oitavem [15], and Arai and Eguchi [16], capture the elementary functions alone, and not in a hierarchy of classes, using various forms of predicative recursion. Leivant [17], captures $\mathcal{E}^3$ using an extension to higher types of ramified recurrence.

V. Marion's diagonalization operator

In [10], the classes of functions $\mathcal{I}_k$ ($k = 0, 1, \ldots$) are defined on the cartesian product of natural numbers. The class $\mathcal{I}_0$ is defined starting from a finite set of linear functions and closing it by composition and *flat recursion*, defined as follows:

$$\begin{cases} f(0,t) &= g(t) \\ f(s+1,t) &= h(s,t) \end{cases}$$

The class $\mathcal{I}_{k+1}$ contains all the functions defined in $\mathcal{I}_k$ and is closed by flat recursion and by (a version of) predicative recursion and composition. It is proved that each class $\mathcal{I}_k$ is the class of the functions computable within polynomial-time bound n^k; thus, $\bigcup_{k<\omega} \mathcal{I}_k$ is the class of polynomial time computable functions. The proof works by induction: if a function g belongs to $\mathcal{I}_0$, then a function F_k computing g^{n^k} can be defined as follows:

$$\begin{cases} F_0(0,x,y) &= g(y) \\ F_{k+1}(0,x,y) &= y \\ F_{k+1}(s+1,x,y) &= F_k(x,x,F_{k+1}(s,x,y)) \end{cases}$$

F_k belongs to $\mathcal{I}_k$, for all k. Note how the variable x is used in order to jump from each class $\mathcal{I}_k$ to class $\mathcal{I}_{k+1}$. In the last line of the definition, the number of computations of the function F_{k+1} is set by using the first occurrence of x, and the information about how many times all the remaining functions $F_k, \ldots, F_0$ have to be computed is given by the second occurrence of x itself. Now the *small jump operator* is defined, in order to jump from every class to the next one:

$$\begin{cases} \Delta[F_k](0,x,y) &= y \\ \Delta[F_k](r+1,x,y) &= F_k(x,x,\Delta[F_k](r,x,y)) \end{cases}$$

It is proved in [10] that $\Delta[F_k](r,x,y) = F_{k+1}(r,x,y)$. If the parameter k assumes the role of a variable, Δ^ω is defined as follows:

$$\begin{cases} \Delta^\omega[g](0,n,x,y) & = & g(y) \\ \Delta^\omega[g](r+1,0,x,y) & = & y \\ \Delta^\omega[g](r+1,n+1,x,y) & = & \\ \qquad = \Delta^\omega[g](r,x,x,\Delta^\omega[g](r+1,n,x,y)) & & \end{cases}$$

The operator Δ^ω, which is not a polynomial function, computes all the polynomial-time computable functions. Contrasting with our approach, no enumerating functions are used; starting from a function in I_0, the small jump operator defines a chaining of "growing" functions (w.r.t. the time complexity) and this chaining is used to jump out of polynomial class. The problem of how to define functions with higher time-complexity is not addressed.

VI. Conclusions and further work

In our previous work, we have used a version of safe recursion and constructive diagonalization to define a hierarchy of classes of programs $\mathcal{T}_\lambda$, with $0 \leq \lambda < \omega^\omega$. Each finite level of the hierarchy characterizes the register machines computing their output within time $O(n^k)$; using the natural definition of structured ordinals, and combining it with the diagonalization operator, the transfinite levels of the hierarchy characterize the classes of register machine computing their output within time bounded by the slow-growing function $B_\lambda(n)$, up to the machines with exponential-time complexity. In this paper, we have given a hint of how to extend our hierarchy further, reaching the class $\mathcal{E}^3$ at level ϵ_0.

While predicative recursion has been studied thoroughly, we feel that the diagonalization operator as presented in this work deserves a more accurate analysis. In particular, we believe that it is as predicative as the recursion, and that it could be used to stretch the hierarchy of programs in order to capture the low Grzegorczyk classes above the elementary level.

References

[1] A. Grzegorczyk, Some classes of recursive functions. Rozprawy Matematyczne, vol. IV, 1953.

[2] H. E. Rose, Subrecursion: functions and hierarchies. Clarendon press, Oxford, 1984.

[3] D. Leivant, "Stratified functional programs and computational complexity," in Proceedings of the 20th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, (POPL'93), Charleston, 1993, pp. 325–333.

[4] K.-H. Niggl, "The μ-measure as a tool for classifying computational complexity," Archive for Mathematical Logic, vol. 39, no. 7, 2000, pp. 515–539.

[5] S. Bellantoni and K.-H. Niggl, "Ranking primitive recursion: the low Grzegorczyk classes revisited," SIAM Journal on Computing, vol. 29, no. 2, 1999, pp. 401–4015.

[6] S. Bellantoni and S. Cook, "A New Recursion-Theoretic Characterization Of The Polytime Functions," Computational Complexity, vol. 2, 1992, pp. 97–110.

[7] D. Leivant, Predicative recurrence and computational complexity I: word recurrence and polytime, in Feasible Mathematics II, P.Clote and J.Remmel (eds). Birkauser, 1994, pp. 320–343.

[8] H. Simmons, "The realm of primitive recursion," Arch.Math. Logic, vol. 27, no. 2, 1988, pp. 177–188.

[9] E. Covino and G. Pani, A Slow-growing Hierarchy of Time-bounded Programs, in Advances in Intelligent Systems: Reviews' Book Series, Vol. 1. IFSA Publishing, Barcelona, Spain, 2017, pp. 151–171.

[10] J. Marion, "On tiered small jump operators," Logical Methods in Computer Science, vol. 5, no. 1, 2009.

[11] E. Covino and G. Pani, "A Specialized Recursive Language for Capturing Time-Space Complexity Classes," in The Sixth International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking, (COMPUTATION TOOLS 2015), Nice, France, 2015, pp. 8–13.

[12] M. Fairtlough and S. Weiner, Hierarchies of provably recursive functions, in Handbook of Proof theory, B. Samuel (ed), Studies in logic and the foundations of mathematics, vol. 137. Elsevier, Amsterdam, 1998, chapter 3, pp. 149–207.

[13] S. Caporaso, G. Pani, and E. Covino, "A predicative approach to the classification problem," Journal of Functional Programming, vol. 11, no. 1, 2001, pp. 95–116.

[14] N. Cutland, Computability: an introduction to recursive function theory. Cambridge university press, Cambridge, 1980.

[15] I. Oitavem, "New recursive characterization of the elementary functions and the functions computable in polynomial space," Revista Matematica de la Univaersidad Complutense de Madrid, vol. 10, no. 1, 1997, pp. 109–125.

[16] T. Arai and N. Eguchi, "A new function algebra of EXPTIME functions by safe nested recursion," ACM Transactions on Computational Logic, vol. 10, no. 4, 2009, pp. 1–19.

[17] D. Leivant, "Ramified recurrence and computational complexity III: higher type recurrence and elementary complexity," Annals of Pure and Applied Logic, vol. 96, no. 1–3, 1999, pp. 209–229.

 ISBN: 978-1-61208-613-2

HPC–Bench:

A Tool to Optimize Benchmarking Workflow for High Performance Computing

Gianina Alina Negoita

Department of Computer Science
Iowa State University
Ames, Iowa, USA
Horia Hulubei National Institute
for Physics and Nuclear Engineering
76900 Bucharest-Magurele, Romania
Email: alina@iastate.edu

Glenn R. Luecke

Department of Mathematics
Iowa State University
Ames, Iowa, USA
Email: grl@iastate.edu

Shashi K. Gadia
and
Gurpur M. Prabhu

Department of Computer Science
Iowa State University
Ames, Iowa, USA
Email: gadia@iastate.edu
Email: prabhu@iastate.edu

***Abstract*—HPC–Bench is a general purpose tool to optimize benchmarking workflow for high performance computing (HPC) to aid in the efficient evaluation of performance using multiple applications on an HPC machine with only a "click of a button". HPC–Bench allows multiple applications written in different languages, multiple parallel versions, multiple numbers of processes/threads to be evaluated. Performance results are put into a database, which is then queried for the desired performance data, and then the R statistical software package is used to generate the desired graphs and tables. The use of HPC–Bench is illustrated with complex applications that were run on the National Energy Research Scientific Computing Center's (NERSC) Edison Cray XC30 HPC computer.**

Keywords–HPC; benchmarking tools; workflow optimization.

I. INTRODUCTION

Today's high performance computers (HPC) are complex and constantly evolving making it important to be able to easily evaluate the performance and scalability of parallel applications on both existing and new HPC computers. The evaluation of the performance of applications can be long and tedious. To optimize the workflow needed for this process, we have developed a tool, HPC–Bench, using the Cyclone Database Implementation Workbench (CyDIW) developed at Iowa State University [1], [2]. HPC–Bench integrates the workflow into CyDIW as a plain text file and encapsulates the specified commands for multiple client systems. By clicking the "Run All" button in CyDIW's graphical user interface (GUI) HPC–Bench will automatically write appropriate scripts and submit them to the job scheduler, collect the output data for each application and then generate performance tables and graphs. Using HPC–Bench optimizes the benchmarking workflow and saves time in analyzing performance results by automatically generating performance graphs and tables. Use of HPC–Bench is illustrated with multiple MPI and SHMEM applications [3], which were run on the National Energy Research Scientific Computing Center's (NERSC) Edison Cray XC30 HPC computer for different problem sizes and for different number of MPI processes/SHMEM processing elements (PEs) to measure their performance and scalability.

There are tools similar to HPC–Bench, but each of these tools has been designed to only run specific applications and measure their performance. For example, ClusterNumbers [4] is a public domain tool developed in 2011 that automates the processor benchmarking HPC clusters by automatically analyzing the hardware of the cluster and configuring specialized benchmarks (HPC Challenge [5], IOzone [6], Netperf [7]). ClusterNumbers, the NAS Parallel Benchmarks [8] and the other benchmarking software are designed to only run and give performance numbers for particular benchmarks, whereas HPC–Bench is designed for easy use with any HPC application and to automatically generate performance tables and graphs. PerfExpert [9] is a tool developed to detect performance problems in applications running on HPC machines. Since it is designed to detect performance problems, PerfExpert is different from HPC–Bench.

The objective of this work is to develop an HPC benchmarking tool, HPC–Bench, as described above and then demonstrate its usefulness for a complex example run on NERSC's Edison Cray XC30. This paper is structured as follows: Section II describes the design of the HPC–Bench tool, which is divided in five Parts. Section III describes the complex example mentioned above. Section IV contains our conclusions.

II. TOOL DESIGN

A simple definition of a workflow is the repetition of a series of activities or steps that are necessary to complete a task. The scientific HPC workflow takes in inputs, e.g., input data, source codes, scripts and configuration files, runs the applications on an HPC cluster and produces outputs that might include visualizations such as tables and graphs. Figure 1 shows a typical example for the scientific HPC workflow diagram.

Scientific HPC workflows are a means by which scientists can model and rerun their analysis. HPC–Bench was designed to optimize the evaluation of the performance of multiple applications. HPC–Bench was implemented using the public domain workbench called Cyclone Database Implementation Workbench (CyDIW). CyDIW was used to develop HPC–Bench for the following reasons:

 ISBN: 978-1-61208-613-2

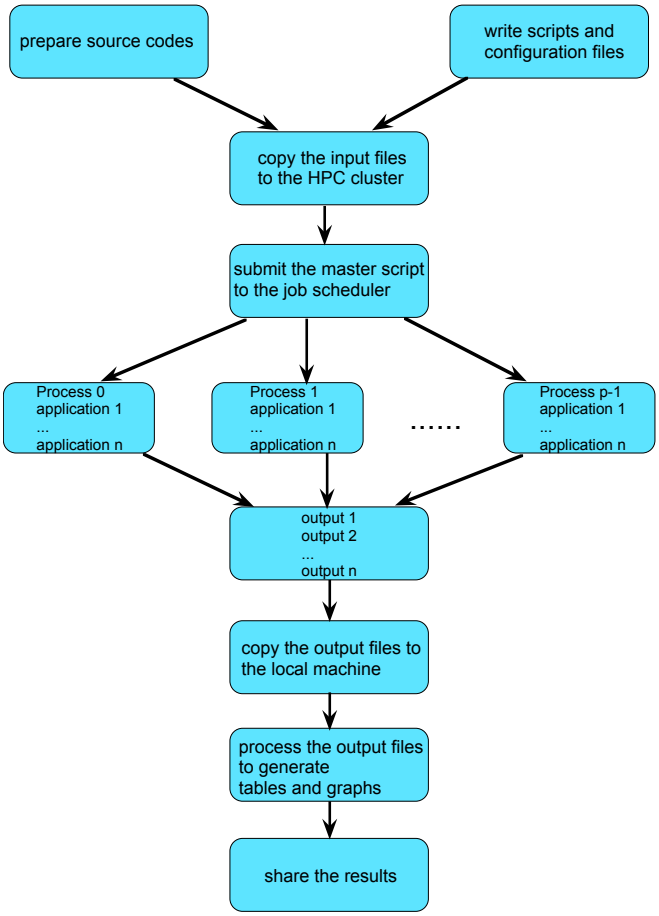

Figure 1. An example for the scientific HPC workflow using n applications that are run on p processes.

- It is easy-to-use, portable (Mac OS, Linux, Windows platforms) and freely available [2].
- It has existing command-based systems registered as clients. The clients used for HPC–Bench are the OS, the open source R environment and the Saxon XQuery engine.
- It has its own scripting language, which includes variables, conditional and loop structures, as well as comments used for documentation, instructions and execution suppression.
- It has a simple and easy-to-use GUI that acts as an editor and a launchpad for execution of batches of CyDIW and client commands.

HPC–Bench uses CyDIW's GUI and database capabilities for managing performance data and contains about 1,000 lines of code. HPC–Bench consists of the following five Parts with illustrations taken from the example described in Section III:

Part 1: XML schema design. An XML schema, known as an XML Schema Definition (XSD), describes the structure of an XML document, i.e., rules for data content. Elements are the main building blocks that contain data, other elements and attributes. Each element definition within the XSD must have a 'name' and a 'type' property. Valid data values for an element in the XML document can be further constrained using the 'default' and the 'fixed' properties. XSD also dictates which subelements an element can contain, the number of instances an element can appear in an XML document, the name, the type and the use of an attribute, etc. The graphical XML schema for this work was created and edited using Altova XMLSpy, see Figure 2. Note the element 'HPC_EXP' contains a sequence of unlimited 'Test' elements, each 'Test' element contains a sequence of 3 'Message' elements, each 'Message' element contains a sequence of 12 'Implementation' elements, each 'Implementation' element contains a choice of unlimited number of 'Process_Rank' elements or 9 'Num_Processes' elements. Each 'Process_Rank' and 'Num_Processes' elements contain a sequence of 'avg', 'max', 'median', 'min' and 'standard_deviation' elements. When using a 'sequence' compositor in XSD, the child elements in the XML document must appear in the order declared in XSD. When using a 'choice' compositor in XSD, only one of the child elements can appear in the XML document. In this work, 'Process_Rank' element will appear in the XML document for the first 'Test' element and 'Num_Processes' otherwise. 'Test' elements stand for applications, 'Message' elements stand for problem sizes, 'Implementation' elements stand for parallel versions, 'Process_Rank' elements stand for process' rank, 'Num_Processes' elements stand for number of MPI processes/SHMEM PEs, while 'avg', 'max', 'median', 'min' and 'standard_deviation' elements stand for statistical timing, respectively.

Part 2: A password-less login to the HPC cluster was implemented. Next, HPC–Bench writes scripts for the submission of the batch jobs. One script is created for each application in a loop and a master script. The master script sets up the environment variables and calls the scripts for each application. This is accomplished by doing the following:

- Use CyDIW's loop structure, *foreach*, to loop through each application.
- Use CyDIW's build-in functions: *createtxt*, *open*, *append*, *appendln*, *appendfile* and *close* to create scripts as text files.
- Use the OS client system registered in CyDIW to copy the files to the HPC cluster.

Part 3: HPC–Bench submits the batch job for execution on the HPC cluster and waits for the job to finish. Suspending the HPC–Bench execution is accomplished by doing the following:

- Launch the job.
- Store its id in a variable.
- Sleep until the 'qstat' command fails, by simply checking the exit status of the 'qstat' command. Once the job is completed, it is no longer displayed by the 'qstat' command.

HPC–Bench next copies the output text files from the HPC cluster to the local machine and converts them to a single written XML file (shown in Figure 3) that follows the XML schema design from Figure 2. An 'awk' script parses the output text files, then a 'shell' script uses the parsed data to create and write the XML file. The XML file is then validated against the XML schema. For example, the 'type' property for an element in XSD must correspond to the correct format of its value in the XML document, otherwise this will cause a validation error when a validating parser attempts to parse the data from the XML document.

 ISBN: 978-1-61208-613-2

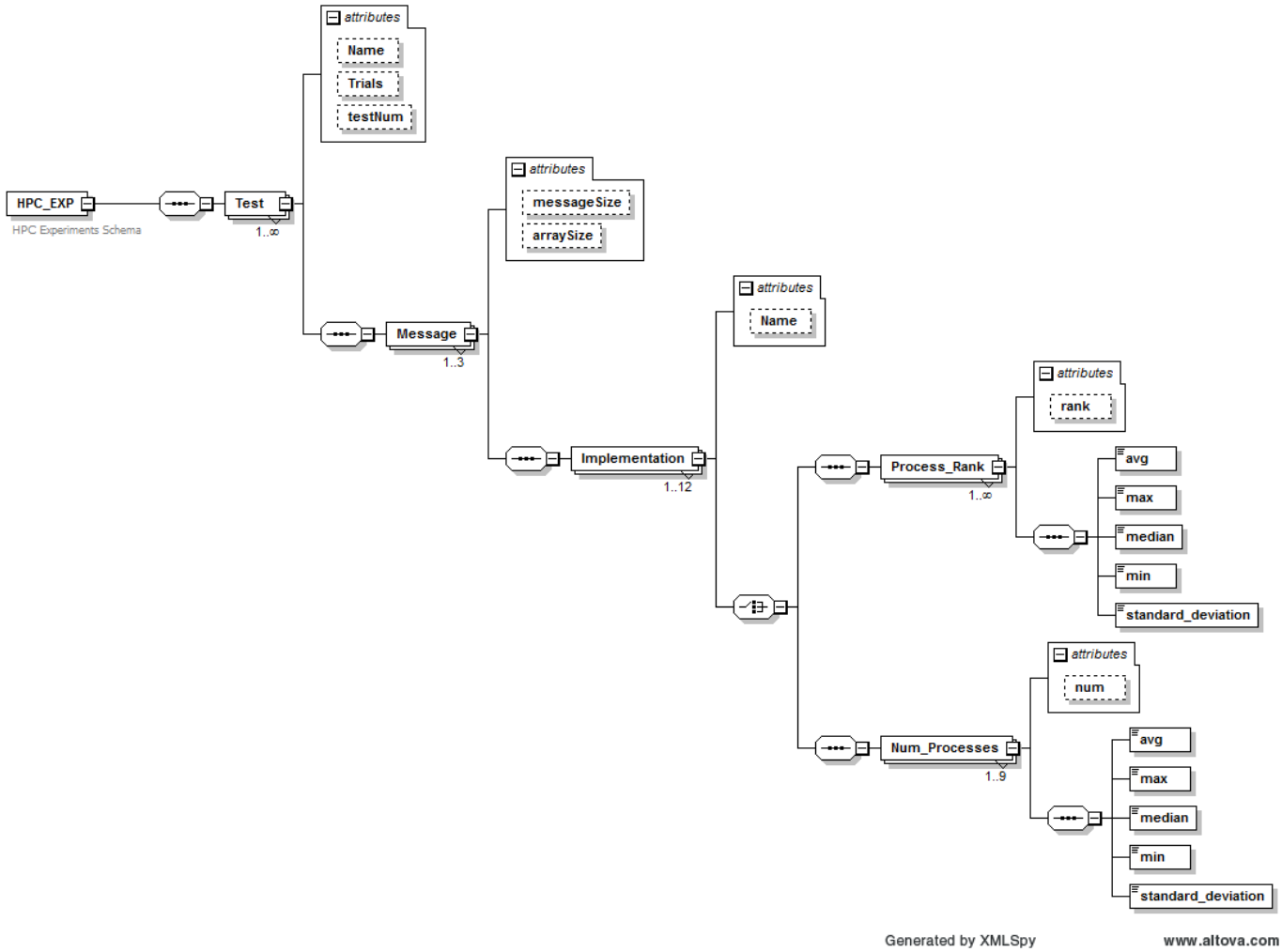

Figure 2. Graphical XML schema using Altova XMLSpy.

```
1  <HPC_EXP xsi:noNamespaceSchemaLocation="HPCExp.SKG.02.xsd" xmlns:xsi="http://www.w3.org/2001/
       XMLSchema-instance">
2      <Test Name="Accessing Distant Messages" Trials="256" testNum="1">
3          <Message messageSize="8 bytes" arraySize="1">
4              <Implementation Name="shmem_get">
5                  <Process_Rank rank="1">
6                      <avg>7.23570582569762599E-4</avg>
7                      <max>9.7059558517284452E-3</max>
8                      <median>6.103706788837984O6E-4</median>
9                      <min>4.4106622240733028 6E-4</min>
10                     <standard_deviation>8.63328421202984395E-4</standard_deviation>
11                 </Process_Rank>
12                 <Process_Rank rank="2">
13                     <avg>3.37445823354852112E-3</avg>
14                     <max>1.40903790087463562E-2</max>
15                     <median>3.11745106205747616E-3</median>
16                     <min>2.52269887546855472E-3</min>
17                     <standard_deviation>1.407381050750595E-3</standard_deviation>
18                 </Process_Rank>
19                 ... data for other ranks, implementations and messages...
20             </Implementation>
21         </Message>
22     </Test>
23     <Test Name="Circular Right Shift" Trials="256" testNum="2">
24         <Message messageSize="8 bytes" arraySize="1">
25             <Implementation Name="shmem_get">
26                 <Num_Processes num="2">
27                     <avg>7.08220533111203585E-4</avg>
28                     <max>1.12190753852561432E-2</max>
29                     <median>6.097459391920033 27E-4</median>
30                     <min>4.19825072886297339E-4</min>
31                     <standard_deviation>9.3970636331058724E-4</standard_deviation>
32                 </Num_Processes>
33                 ... data for other number of processes, implementations,
34                 messages and Tests ...
35     </Test>
36 </HPC_EXP>
```

Figure 3. The XML file containing the output data validated against the XSD from Figure 2.

Part 4: HPC–Bench then queries the XML file for the desired performance data using the XQuery language to generate

- performance tables

and

- the XML input files to the R statistical package that will be used to generate various graphs.

Queries were declared as string variables in CyDIW and then run. Nested *foreach* command was used to iterate through applications 2 to 5 and through different problem/message sizes. Each output generated by the queries was directed to an XML file, see Figure 4.

```
1  // Loop through each Test from 2-5;
2  $CyDB:> foreach $$j in [2, 5]
3  {
4      // Loop through each message size: 8 bytes, 10 Kbytes and 1 Mbyte;
5      $CyDB:> foreach $$k in [1, 3] {
6          $CyDB:> set $$queryRatioTest$$j[$$k] := ...
7          $CyDB:> run $Saxon $$queryRatioTest$$j[$$k] out >> output_tableRatio_Test$$j_$$
                   messageSize2[$$k].xml;
8      }
9  }
```

Figure 4. Example setting the queries as variables and running the queries.

 ISBN: 978-1-61208-613-2

For the first application, we queried the average of the median times over all the ranks for each problem/message size and for each parallel version/implementation. See Figure 5 for generating a performance table for application 1. For the other applications we queried the median times for each run (specified by the number of processes used) for each problem/message size and for each parallel version/implementation. See Figure 6 for producing performance tables for applications 2 to 5.

```
$Saxon:>
<Test1TABLE1Ratios xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <table border="1" >
        {
        let $a := doc("ComS363/Final_Project/input.MPI3.xml")//Test[@testNum="1"]
        return
        <tr> <td>Message Size</td>
            <td >{$a/Message[@messageSize="8 bytes"]/Implementation[@Name="shmem_get"]/
                  @Name/string()}</td>
            <td >{$a/Message[@messageSize="8 bytes"]/Implementation[@Name="mpi_get"]/
                  @Name/string()}</td>
            <td >ratio1</td>
            <td >{$a/Message[@messageSize="8 bytes"]/Implementation[@Name="shmem_put"]/
                  @Name/string()}</td>
            <td >{$a/Message[@messageSize="8 bytes"]/Implementation[@Name="mpi_put"]/
                  @Name/string()}</td>
            <td >ratio2</td>
            <td >{$a/Message[@messageSize="8 bytes"]/Implementation[@Name="mpi_send_recv"
                ]/@Name/string()}</td>
            <td >ratio3</td>
        </tr>
        }
        {
        let $a := doc("ComS363/Final_Project/input.MPI3.xml")//Test[@testNum="1"]
        for $x in $a//@messageSize
        let $i := $a/Message[@messageSize=$x]/Implementation[@Name='shmem_get']//median
        let $j := $a/Message[@messageSize=$x]/Implementation[@Name='mpi_get']//median
        let $k := $a/Message[@messageSize=$x]/Implementation[@Name='shmem_put']//median
        let $l := $a/Message[@messageSize=$x]/Implementation[@Name='mpi_put']//median
        let $m := $a/Message[@messageSize=$x]/Implementation[@Name='mpi_send_recv']//
                  median
        return
        <tr>
            <td> {$x/string()} </td>
            <td>{ round(avg($i) * 10000) div 10000.0} </td>
            <td>{ round(avg($j) * 10000) div 10000.0} </td>
            <td >{round(avg($j) div avg($i) * 100) div 100.0}</td>
            <td>{ round(avg($k) * 10000) div 10000.0} </td>
            <td>{ round(avg($l) * 10000) div 10000.0} </td>
            <td >{round(avg($l) div avg($k) * 100) div 100.0}</td>
            <td>{ round(avg($m) * 10000) div 10000.0} </td>
            <td >{round(avg($m) div avg($i) * 100) div 100.0}</td>
        </tr>
        }
    </table>
</Test1TABLE1Ratios>;
```

Figure 5. Query that gives a performance table for application 1.

```
$CyDB:> foreach $$j in [2, 5] // Loop through each Test from 2-5;
{
$CyDB:> set $$queryRatio_8bytes[$$j] :=
<Test$$j_TABLE$$j_Ratios_8bytes $$namespace>
    <table border="1" >
        {
        let $a := $$xmldoc//Test[@testNum="$$j"]/Message[@messageSize="8 bytes"]
        return
        <tr> <td >Message Size</td > <td >8 bytes </td >
            <tr> Number of Processes </tr>
            <td >{$a/Implementation[@Name="shmem_get"]/@Name/string()}</td>
            <td >{$a/Implementation[@Name="mpi_get"]/@Name/string()}</td>
            <td >ratio1</td>
            <td >{$a/Implementation[@Name="shmem_put"]/@Name/string()}</td>
            <td >{$a/Implementation[@Name="mpi_put"]/@Name/string()}</td>
            <td >ratio2</td>
            $$implementationRatioString1[$$j]
        </tr>

        }
        {
        let $a := $$xmldoc//Test[@testNum="$$j"]/Message[@messageSize="8 bytes"]
        for $x in $a/Implementation[@Name='shmem_get']//@num
        let $i := $a/Implementation[@Name='shmem_get']/Num_Processes[@num=$x]/median
        let $j := $a/Implementation[@Name='mpi_get']/Num_Processes[@num=$x]/median
        let $k := $a/Implementation[@Name='shmem_put']/Num_Processes[@num=$x]/median
        let $l := $a/Implementation[@Name='mpi_put']/Num_Processes[@num=$x]/median
        return
        <tr>
            <td> {$x/string()} </td>
            <td>{ round($i * 10000) div 10000.0} </td>
            <td>{ round($j * 10000) div 10000.0} </td>
            <td>{ round($j div $i * 100) div 100.0} </td>
            <td>{ round($k * 10000) div 10000.0} </td>
            <td>{ round($l * 10000) div 10000.0} </td>
            <td>{ round($l div $k * 100) div 100.0} </td>
            $$implementationRatioString2[$$j]
        </tr>
        }
    </table>
</Test$$j_TABLE$$j_Ratios_8bytes>;
$CyDB:> set $$queryRatio_10Kbytes[$$j] :=....
...
$CyDB:> set $$queryRatio_1Mbyte[$$j] :=....
}

// Produce the tables for Tests 2-5 for all message sizes;
// Loop through each Test from 2-5;
$CyDB:> foreach $$j in [2, 5]
{
$CyDB:> run $$prefix $$queryRatio_8bytes[$$j] out >> output_tableRatio_Test$$j_8bytes.xml;
$CyDB:> run $$prefix $$queryRatio_10Kbytes[$$j] out >> output_tableRatio_Test$$j_10Kbytes.xml
      ;
$CyDB:> run $$prefix $$queryRatio_1Mbyte[$$j] out >> output_tableRatio_Test$$j_1Mbyte.xml;
}
```

Figure 6. Query that gives performance tables for applications 2 to 5.

The database was then queried for the data needed to generate the performance graphs. Figure 7 shows the query that gives the median times for all the parallel versions/implementations for 8-byte messages for application 2. The XML file containing the performance data obtained by this query is shown in Figure 8.

```
$CyDB:> set $$query_plot_8bytes[2] :=
<Test$$j_plot$$j_8bytes $$namespace>
    {
    let $a := $$xmldoc//Test[@testNum="$$j"]/Message[@messageSize="8 bytes"]
    for $x in $a/Implementation[@Name='shmem_get']//@num
    return
    <Num_Processes>
        {
        <num_pes> {$x/string()} </num_pes>,
        <shmem_get> { round($a/Implementation[@Name='shmem_get']/Num_Processes[@num=$x]/
                median * 10000) div 10000.0} </shmem_get>,
        <mpi_get>{ round($a/Implementation[@Name='mpi_get']/Num_Processes[@num=$x]/median *
                10000) div 10000.0} </mpi_get>,
        <shmem_put>{ round($a/Implementation[@Name='shmem_put']/Num_Processes[@num=$x]/median
                * 10000) div 10000.0} </shmem_put>,
        <mpi_put>{ round($a/Implementation[@Name='mpi_put']/Num_Processes[@num=$x]/median *
                10000) div 10000.0} </mpi_put>,
        $$implementationString[$$j]
        }
    </Num_Processes>
    }
</Test$$j_plot$$j_8bytes>
;
$CyDB:> run $Saxon $$query_plot_8bytes[2] out >> output_plot_Test2_8bytes.xml;
```

Figure 7. Query that gives the performance data needed to generate the performance graph for 8-byte messages for application 2.

```
<Root>
  <Test2_plot2_8bytes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <Num_Processes>
      <num_pes>2</num_pes>
      <shmem_get>0.0005</shmem_get>
      <mpi_get>0.0113</mpi_get>
      <shmem_put>0.0013</shmem_put>
      <mpi_put>0.0096</mpi_put>
      <mpi_sendrecv>0.0026</mpi_sendrecv>
      <mpi_isend_irecv>0.0037</mpi_isend_irecv>
      <mpi_send_recv>0.0054</mpi_send_recv>
   </Num_Processes>
   <Num_Processes>
      <num_pes>4</num_pes>
      <shmem_get>0.0051</shmem_get>
      <mpi_get>0.0169</mpi_get>
      <shmem_put>0.007</shmem_put>
      <mpi_put>0.0155</mpi_put>
      <mpi_sendrecv>0.0093</mpi_sendrecv>
      <mpi_isend_irecv>0.0076</mpi_isend_irecv>
      <mpi_send_recv>0.0084</mpi_send_recv>
   </Num_Processes>
.......
  </Test2_plot2_8bytes>
</Root>
```

Figure 8. The XML file generated by the query above for application 2.

Part 5: HPC–Bench uses R to generate the performance graphs. This is accomplished by first converting the XML files generated by the queries for graphs from Part 4 (see Figure 8 as an example) to R dataframes and then setting up the plotting environment, e.g., the size of the graphs, the style of the X and Y axes, graph labels, colors, legends, etc.

The first step for generating the performance graphs is to install the "XML", "plyr", "ggplot2", "gridExtra" and "reshape2" R packages and load them in R. The "plyr" package is used to convert the XML file to a dataframe. Next, HPC–Bench reads the XML file into an R tree, i.e., R-level XML node objects using the *xmlTreeParse()* function.

 ISBN: 978-1-61208-613-2

Then HPC–Bench uses the *xmlApply()* function for traversing the nodes (applies the same function to each child of an XML node). $function(node)$ $xmlSApply(node, xmlValue)$ does the initial processing of an individual *Num_Processes* node, where *xmlValue()* returns the text content within an XML node. This function must be called on the first child of the root node, e.g., $xmlSApply(doc[[1]], xmlValue)$. All the *Num_Processes* nodes are processed with the command $xmlSApply(doc[[1]], function(x)\ xmlSApply(x, xmlValue))$. The result is a character matrix whose rows are variables and whose columns are records. After transposing this matrix, it is converted to a dataframe. As an example, see Figure 9 that generates the dataframe shown in Table I for application 2. This completes working with XML files and the rest is R programming.

```
# Nodes traversing function
function(node) xmlSApply(node, xmlValue)
doc = xmlRoot(xmlTreeParse(inputFile.xml)
numLoop = xmlSize(doc[[1]])
tmp = xmlSApply(doc[[1]], function(x) xmlSApply(x, xmlValue))
tmp = t(tmp) # transpose matrix
df = as.data.frame(matrix(as.numeric(tmp), numLoop))
names(df)<- c("Number Processes", "shmem_get", "mpi_get", "shmem_put", "mpi_put", "
        mpi_sendrecv", "mpi_isend_irecv", "mpi_send_recv")
```

Figure 9. Code to convert an XML file to an R dataframe.

TABLE I. THE R DATAFRAME GENERATED WITH THE CODE FROM FIGURE 9 FOR 8-BYTE MESSAGE SIZE FOR APPLICATION 2.

	Num Proc	*shmem get*	*mpi get*	*shmem put*	*mpi put*	*mpi send--recv*	*mpi isend irecv*	*mpi send recv*
1	2	0.0005	0.0113	0.0013	0.0096	0.0026	0.0037	0.0054
2	4	0.0051	0.0169	0.0070	0.0155	0.0093	0.0076	0.0084
3	8	0.0046	0.0178	0.0084	0.0171	0.0118	0.0106	0.0125
4	16	0.0056	0.0246	0.0088	0.0250	0.0124	0.0115	0.0137
5	32	0.0048	0.0289	0.0088	0.0269	0.0142	0.0126	0.0113
6	64	0.0053	0.0357	0.0112	0.0329	0.0144	0.0134	0.0160
7	128	0.0054	0.0494	0.0122	0.0378	0.0165	0.0190	0.0215
8	256	0.0057	0.0518	0.0120	0.0502	0.0207	0.0225	0.0232
9	384	0.0093	0.0584	0.0198	0.0540	0.0223	0.0224	0.0247

After obtaining the R dataframes, HPC–Bench sets up the plotting environment as follows:

- Use the "ggplot2","gridExtra" and "reshape2" R packages to create graphs and put multiple graphs on one panel.
- Write a function to create minor ticks and then write another function to mirror both axes with ticks.
- Set and update a personalized theme: *theme_set(theme_bw()), theme_update(. . .)*.
- For each application, plot the dataframe for each problem/message size using the *ggplot()* function with personalized options. See Figure 10.

```
p <- ggplot(data=df.melted, aes(x='Number Processes', y=value, group=variable, shape=factor(
        variable), color=variable))
p <- p + geom_line(aes(linetype=variable)) + geom_point(fill = "white", size = 2.5)
p <- p + geom_line(aes(linetype=variable)) + geom_point(fill = "white", size = 2.5)
p <- p + scale_colour_manual(messageSize[c(i)], values=c("red", "red", "blue", "blue", "
        brown4", "darkgreen", "green"), labels=c("SHMEM get", "MPI get","SHMEM put", "MPI
        put", "MPI sendrecv", "MPI isend&irecv", "MPI send&recv"))
```

Figure 10. Code that generates a plot using the df dataframe.

For each application and for each problem/message size, HPC–Bench plots the desired timing data for all versions/implementations. Next, for each application, HPC–Bench places the three plots for different problem/message sizes (p1, p2 and p3) into one panel using *gtable* to generate a graph, that is then printed to PDF format, see Figure 11. At the end of the HPC–Bench execution, performance graphs are displayed for all applications in popup windows. Figures 14 and 15 illustrate this.

```
ge <- gtable:::rbind_gtable(p1, p2, "first")
g <- gtable:::rbind_gtable(ge, p3, "first")
grid.newpage()
# grid.draw(ge) # draw 2 figures
grid.draw(g) # draw 3 figures, show the plot
# Print to pdf using pdf and plot
pdf(outputFile)
plot(g)
dev.off()
```

Figure 11. Code that places 3 plots into one panel.

Figure 12 shows the HPC workflow diagram for HPC–Bench. The blue boxes are components of the HPC workflow, which include input data and output data to manage, as well as source codes, scripts and configuration files for the system. The red boxes show the portions of the HPC workflow controlled by HPC–Bench.

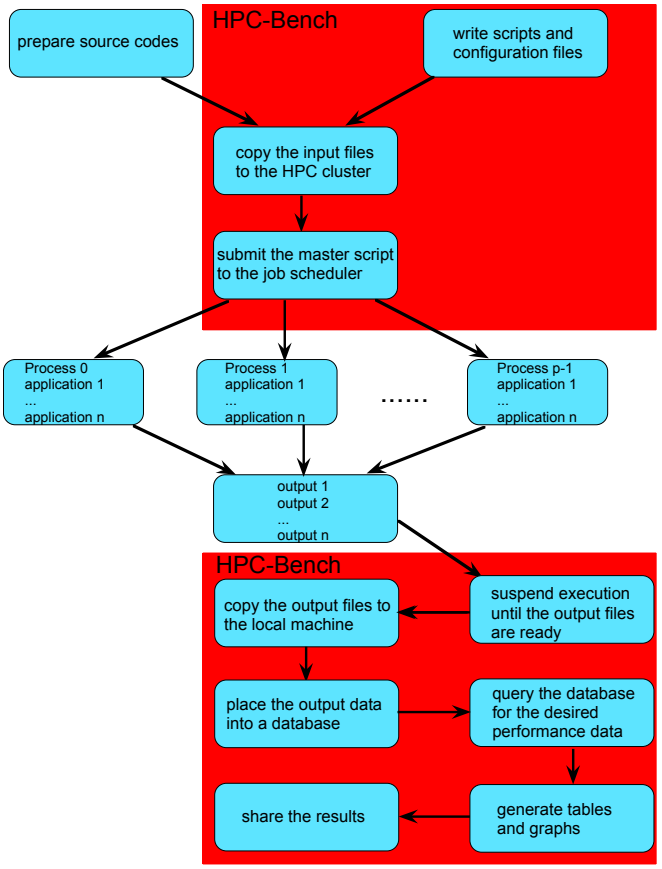

Figure 12. HPC workflow diagram for HPC–Bench.

Since the output processing part cannot begin until all the runs are complete, HPC–Bench suspends execution until all the output data is available. HPC–Bench then puts the output data into a database and queries it for the desired results.

III. EXAMPLE USING HPC–BENCH

In this section, we illustrate how HPC–Bench can be used in a complex benchmarking environment. The example and the benchmarking environment information come from [3]. The

benchmark tests used for this example were: accessing distant messages, circular right shift, gather, broadcast, and all-to-all. Each test has several parallel versions, which use: MPI *get*, *put*, blocking and non-blocking *sends*/*receives*, *gather*, *broadcast* and *alltoall* routines as well as the SHMEM *get*, *put*, *broadcast* and *alltoall* routines.

The NERSC's Edison Cray XC30 with the Aries interconnect was used for benchmarking. Edison has 5576 XC30 nodes with 2 Intel Xeon E5-2695v2 12-chip processor for a total of 24 cores per node. There are 30 cabinets and each cabinet consists of 192 nodes. Cabinets are interconnected using the Dragonfly topology with 2 cabinets in a single group.

For this example, 2 cabinets in a single group (2x192 nodes) were reserved. Each application was run with 2 MPI processes/SHMEM PEs per node using message sizes of 8 bytes, 10 Kbytes and 1 Mbyte and 2 to 384 MPI processes/SHMEM PEs.

Use of HPC–Bench is illustrated via CyDIW's GUI, shown in Figure 13. The GUI is intentionally designed to be as simple as possible for ease-of-use: it has a "Commands Pane", an "Output Pane" and a "Console". The "Commands Pane" acts as an editor and a launch-pad for execution of batches of commands, written as text files. The output can be shown in the "Output Pane", directed to files, or displayed in popup windows. The "Output Pane" is an html viewer, but it can display plain text as well. For example, a user can see an html table computed by an XQuery query displayed in the "Output Pane". The html code or the display in an html browser can be viewed without having to get out of the GUI in order to use a text editor or an html browser. The "Console" displays the status and error messages for the commands.

In CyDIW's GUI, click "Open" and then browse to the HPC–Bench file to open HPC–Bench. One can run all the applications from scratch and produce the performance tables and graphs in a "click of a button" by clicking the "Run All" button. HPC–Bench displays one three-panel graph for each application in a popup window. See Figures 14 and 15 as examples for performance graphs produced by HPC–Bench.

Figure 14 shows the median time in milliseconds (ms) versus the process' rank for the accessing distant messages test with 8-byte, 10-Kbyte and 1-Mbyte messages. The purpose of this test is to determine the performance differences of 'sending' messages between 'close' processes and 'distant' processes using SHMEM and MPI routines. The curves represent various implementations of this test using the SHMEM and MPI *get* and *put* routines, as well as the MPI *send*/*receive* routines as shown in the legend. Figure 14 shows that times to access messages within a group of two cabinets on NERSC's Edison Cray XC30 were nearly constant for each implementation, showing the good design of the machine.

Figure 15 shows the median time in milliseconds (ms) versus the number of processes for the circular right shift test with 8-byte, 10-Kbyte and 1-Mbyte messages. In this test, each process 'sends' a message to the right process and 'receives' a message from the left process. The curves represent various implementations of this test using the SHMEM and MPI *get* and *put* routines, as well as the MPI two-sided routines, e.g., *send*/*receive*, *isend*/*ireceive* and *sendrecv* as shown in the legend. Figure 15 shows that all implementations scaled well with the number of processes for all message sizes.

HPC–Bench can be easily modified by clicking the "Edit" button to run only selected applications or to change the number of processes, library version or configuration to run on, as well as to add more queries to do a different performance analysis. Alternatively, one can run parts of HPC–Bench selecting which parts to run and then clicking the "Run Selected" button. This is useful when one would like to produce additional tables and graphs from existing output data without having to rerun the applications.

IV. Conclusion

HPC–Bench is a general purpose tool to minimize the workflow time needed to evaluate the performance of multiple applications on an HPC machine at the "click of a button". HPC–Bench can be used for performance evaluation for multiple applications using multiple MPI processes, Cray SHMEM PEs, threads and written in Fortran, Coarray Fortran, C/C++, UPC, OpenMP, OpenACC, CUDA, etc. Moreover, HPC–Bench can be run on any client machine where R and the CyDIW workbench have been installed. CyDIW is preconfigured and ready to be used on a Windows, Mac OS or Linux system where Java is supported. The usefulness of HPC–Bench was demonstrated using complex applications on a NERSC's Cray XC30 HPC machine.

Acknowledgment

This research used resources of the National Energy Research Scientific Computing Center (NERSC), a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. Personnel time for this project was supported by Iowa State University.

References

[1] X. Zhao and S. K. Gadia, "A Lightweight Workbench for Database Benchmarking, Experimentation, and Implementation," IEEE Transactions on Knowledge and Data Engineering, vol. 24, no. 11, Nov. 2012, pp. 1937–1949, DOI: 10.1109/TKDE.2011.169, ISSN: 1041-4347.

[2] "Cyclone Database Implementation Workbench (CyDIW)," 2012, URL: http://www.research.cs.iastate.edu/cydiw/ [accessed: 2018-01-10].

[3] G. A. Negoita, G. R. Luecke, M. Kraeva, G. M. Prabhu, and J. P. Vary, "The Performance and Scalability of the SHMEM and Corresponding MPI Routines on a Cray XC30," in Proceedings of the 16^{th} International Symposium on Parallel and Distributed Computing (ISPDC 2017) July 3–6, 2017, Innsbruck, Austria. IEEE, Jul. 2017, pp. 62–69, DOI: 10.1109/ISPDC.2017.19, ISBN: 978-1-5386-0862-3.

[4] "ClusterNumbers," 2011, URL: https://sourceforge.net/projects/clusternumbers/ [accessed: 2018-01-10].

[5] "The HPC Challenge Benchmarks," URL: http://icl.cs.utk.edu/hpcc/ [accessed: 2018-01-10].

[6] "IOzone," URL: http://iozone.org/ [accessed: 2018-01-10].

[7] "Netperf," URL: https://hewlettpackard.github.io/netperf/ [accessed: 2018-01-10].

[8] "The NAS Parallel Benchmarks derived from computational fluid dynamics (CFD) applications," URL: www.nas.nasa.gov/publications/npb.html [accessed: 2018-01-10].

[9] M. Burtscher, B. D. Kim, J. Diamond, J. McCalpin, L. Koesterke, and J. Browne, "PerfExpert: An Easy-to-Use Performance Diagnosis Tool for HPC Applications," in Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2010, November 13–19, 2010, New Orleans, LA, USA. ACM/IEEE, Nov. 2010, pp. 1–11, DOI: 10.1109/SC.2010.41.

 ISBN: 978-1-61208-613-2

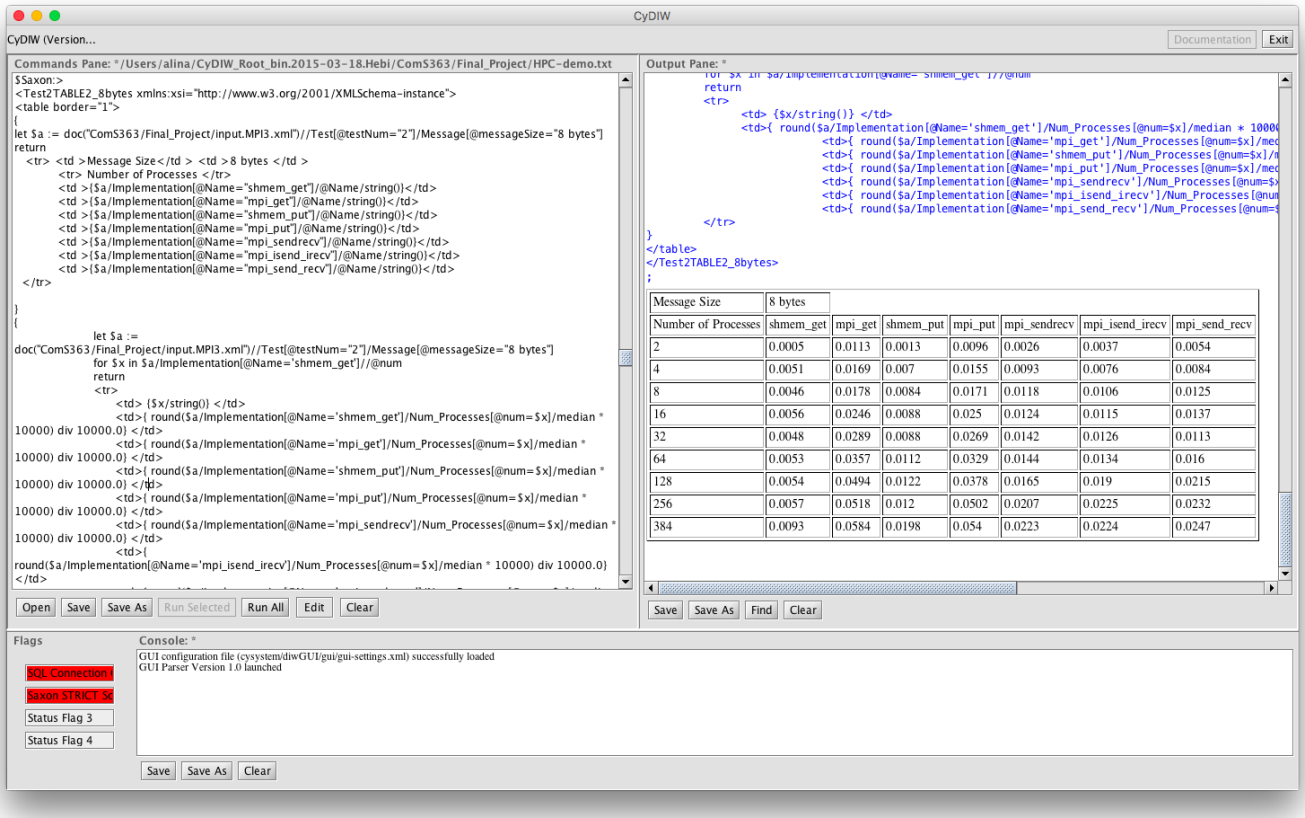

Figure 13. CyDIW's GUI showing the table generated by XQuery for 8-byte message for application 2, containing the same performance data as Table I.

Message Size	8 bytes						
Number of Processes	shmem_get	mpi_get	shmem_put	mpi_put	mpi_sendrecv	mpi_isend_irecv	mpi_send_recv
2	0.0005	0.0113	0.0013	0.0096	0.0026	0.0037	0.0054
4	0.0051	0.0169	0.007	0.0155	0.0093	0.0076	0.0084
8	0.0046	0.0178	0.0084	0.0171	0.0118	0.0106	0.0125
16	0.0056	0.0246	0.0088	0.025	0.0124	0.0115	0.0137
32	0.0048	0.0289	0.0088	0.0269	0.0142	0.0126	0.0113
64	0.0053	0.0357	0.0112	0.0329	0.0144	0.0134	0.016
128	0.0054	0.0494	0.0122	0.0378	0.0165	0.019	0.0215
256	0.0057	0.0518	0.012	0.0502	0.0207	0.0225	0.0232
384	0.0093	0.0584	0.0198	0.054	0.0223	0.0224	0.0247

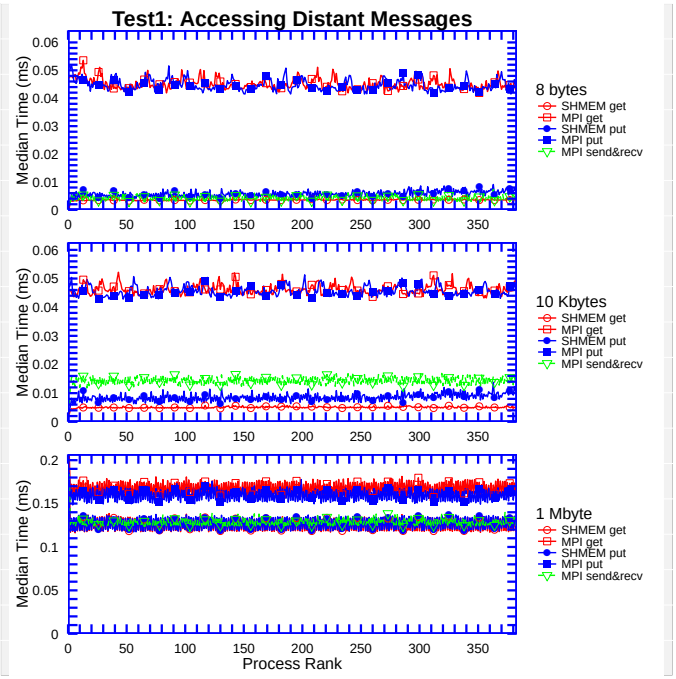

Figure 14. An example of a graph generated by HPC–Bench for application 1, accessing distant messages test.

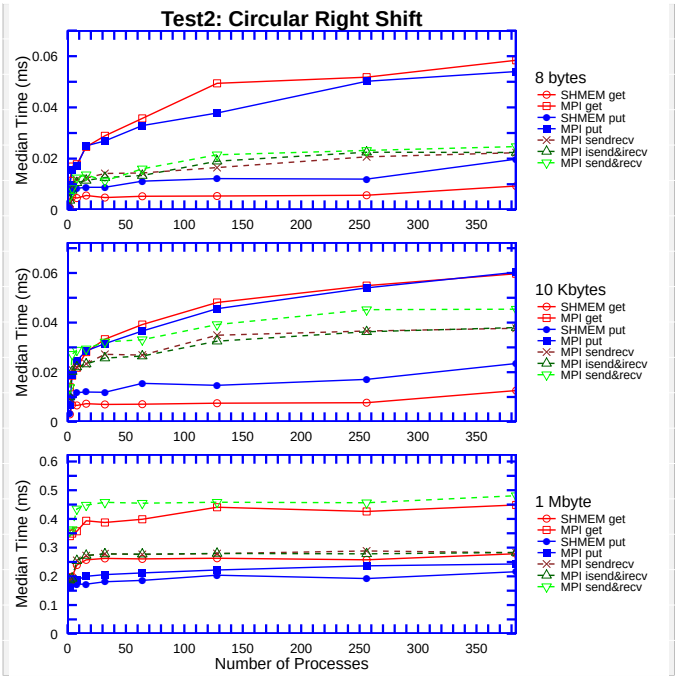

Figure 15. An example of a graph generated by HPC–Bench for application 2, circular right shift test.

License Plates Recognition of Mexican Private Vehicles

Carlos Hiram Moreno, Nicolás Trejo, Martha Soto
Departmento de Ingeniería en Sistemas Computacionales
Tecnológico de Estudios Superiores de Chimalhuacan
Chimalhuacán Estado de México
Email: {carlosmoreno, nicolastrejo, sistemasteschi}
@teschi.edu.mx

Benjamin Moreno-Montiel
Departamento de Ingeniería Eléctrica
Universidad Autónoma Metropolitana
México City
Email: bmm@xanum.uam.mx

***Abstract*— In most of the investigations about the recognition of license plates from different countries it is assumed that they have a white background, without texture patterns and with black characters. Vehicle registration plates in Mexico are different because they have different texture patterns and colors in the background depending on the State of the Republic; that is why the algorithms of recognition of these plates are not always successful. This article proposes an algorithm for the recognition of vehicle registration plates of Mexico considering three phases: A) Normalization and Binarization of the plate, which is achieved by using a threshold factor, which separates dark colors that form letters from the clearings that are at the bottom. B) Characterization C) Modeling of symbols by technics such as Hu's moment, Fourier descriptors and correlation cross factors and D) Classification, where we have used comparisons between different techniques of template matching, Bayesian classifier and Artificial Neural Networks to process images of plates from different states. The results obtained are discussed at the end of the present paper.**

Keywords-license plate recognition; bayesian classifier; artificial neural network; correlation factor; principal component analysis; Hu's moments.

I. INTRODUCTION

Recognition of license plates of vehicles has been investigated throughout the world. We can find some examples of recognition of license plates in countries like Argentina [1], Bangladesh [2], China [3], Egypt [4], India [5], Japan [6], Malaysia [7], among others. Normally, these works consider four phases: 1) get the image of the vehicle 2) plates location inside the image, 3) characters extraction and 4) classification or recognition of characters.

In most algorithms found in literature, it is assumed that the plate has no textured patterns, the background is usually white and the characters black allowing better character recognition. However, in the case of Mexican plates it is not the case. On one hand, they have patterns of texture in the background and on the other hand, each State Government can design their own pattern of background texture; this generates more than 32 different plates, and this number increase with changes in the government administration.

These structures in the vehicle registration plates of Mexico make the traditional algorithms not working properly, mainly in the phase of extraction of characters.

Another important thing to mention is that although each State can design the background of its plates, the dimensions of the plates and letters, as well as their style, must meet with the features that are designated in the official Mexican standard NOM-001-SCT-2-2000; these characteristics are used to recognize the registration.

This work proposes an algorithm that segments each character and recognizes them depending on their characteristics of color and form. To properly segment the characters, most of the background texture patterns were eliminated by using a threshold factor, which separated the dark colors that make up the letters from the light colors that make up the background. Once filtered the background texture, the image of the plate was binarized and both vertical and horizontal histograms were obtained using the technique of projection of profiles, just to obtain the coordinates of the position used to segment characters. When the images of the characters were obtained, we proceeded to model them and characterize them using the techniques of Hu's moment, Fourier Descriptors, and Cross-correlation factor. Data obtained at this stage was used as complementary in the classification phase. Finally, in the stage of classification, techniques of templates, Bayesian classifier and artificial networks neural were used. The results obtained are discussed at the end of the present work.

The article is organized in the following form: in Section 2 is presented the proposal of recognition of license plates. Experiments conducted in the section are presented in Section 3. A discussion of the results obtained is made in Section 4 and the article ends with conclusions in Section 5.

II. PROPOSAL OF VEHICLE LICENSE PLATE RECOGNITION

Automatic Number Plate Recognition (ANPR) presented in Fig. 1, consists essentially of four stages: 1) To get the image of the vehicle using a camera, 2) Extract the image plate, 3) Segment and remove the plate characters and 4) Recognize the characters extracted.

In the present work is only discussed the segmentation and recognition of characters, stages (3) and (4). Images employed contain exclusively the region that conforms to the plate, and has the following characteristics:

- Images of the plates must be obtained between 1 and 1.5 meters of distance between the camera and the registration.
- They must not have lighting variations.
- Images are frontal or near frontal, meaning images must have very small rotation angles.
- No occlusions or considerable physical damages.

 ISBN: 978-1-61208-613-2

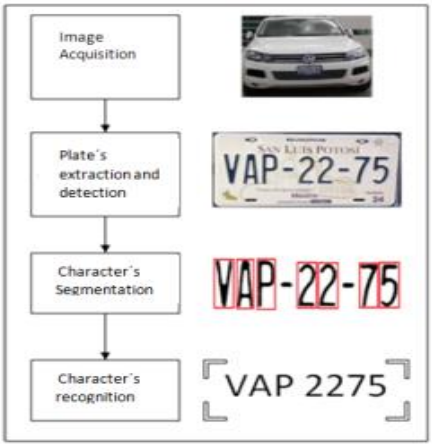

Figure 1. Stages of an ANPR system.

The developed algorithm in this work is presented and consists of: 1) Standardization and binarization of the plate, 2) Segmentation of the characters, 3) Character modeling and 4) Recognition of characters.

A. *Normalization and Binarization of the plate*

Normalization of the images is based on the standard NOM-001-SCT-2-2000, where it is established that dimensions of the plates must be of 2:1, so, all images are resized at 700 ×350 pixels of being binarized. The same standard also establishes the position in which the characters must be collocated inside the plate, as well as the size and the distance between them, as it is shown in Fig. 2.

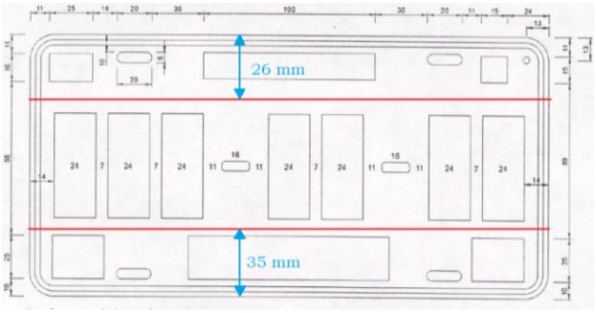

Figure 2. Position of the characters inside the registration.

With this information is eliminated the upper and lower sections of the original image, with the objective to get a section that contains exclusively characters. The result is shown in Fig. 3.

Figure 3. Example of sections elimination upper and lower; (a) original image, (b) image obtained after sections elimination.

Due to the great variety of background colors in plates, before being binarized, it was necessary to separate the texture of the background from the numbers and letters of plates.

It is important to mention that, in the present work, the image colors are represented in the space RGB. In this space, the colors are represented as a linear combination of the vectors base in red, green and blue; the color of a pixel is represented as ϕ=[r, g, b]. Fig. 4 shows the form of the RGB space.

To develop the separation process, we use an experimental threshold by graythresh function defined in Matlab, which chooses the threshold to minimize the interclass variance of the black and white pixels; the threshold (δ) obtained is of 0.73. To separate what is in the magnitude of a vector RGB, we start by performing a comparison to classify a pixel in a region or another, shown in Fig. 4.

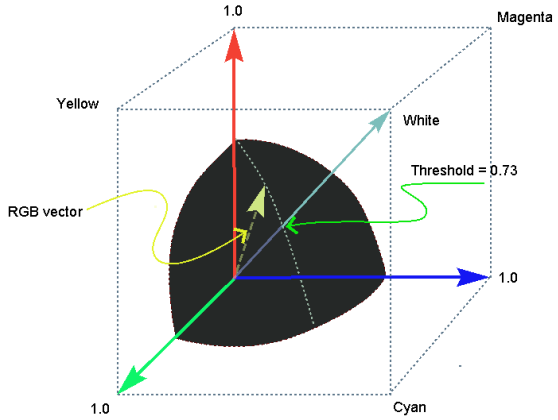

Figure 4. RGB space and threshold of intensity.

In Fig. 4 it is showed the part of the space where the colors from the characters are found. Separation was made using the equation (1).

$$\theta^* = \begin{cases} \vec{1}, & \|\phi\| \leq \delta \\ \vec{0}, & \|\phi\| > \delta \end{cases} \qquad (1)$$

where $\eta^* = 0.73$, $\vec{1} = [1,1,1]$ and $\vec{0} = [0,0,0]$. With the proposed method, it is managed to differentiate the characters; the background was obtained and therefore the possibility of binarizing the images. In Fig. 5 we show two examples of binarized images using the Otsu method [8], which is normally used for gray scales; therefore, when using black and white is also a case for Otsu method.

Figure 5. Example of two images of plates binarized using the Otsu method.

B. *Characters Segmentation*

The horizontal and vertical projection method is used to segment characters [9]; this method has been used for similar investigations in [6][10][11].

Suppose we have an image $\boldsymbol{I}(\boldsymbol{x},\boldsymbol{y})$ with $wide = N$ and $heigth = M$ (considering that $1 \le x \le N$ and $1 \le y \le M$), the horizontal and vertical projections are defined as:

$$P_{hor}(y_0) = \sum_{x=1}^{N} I(x, y_0), \forall y = 1, \dots, M \quad (2)$$

$$P_{ver}(x_0) = \sum_{y=1}^{M} I(x_0, y), \forall x = 1, \dots, N \quad (3)$$

The histograms obtained by equations (2) and (3) allow determining the coordinates of the area of each letter and number in the plates. Fig. 6 shows an example of horizontal and vertical projection of a binarized image.

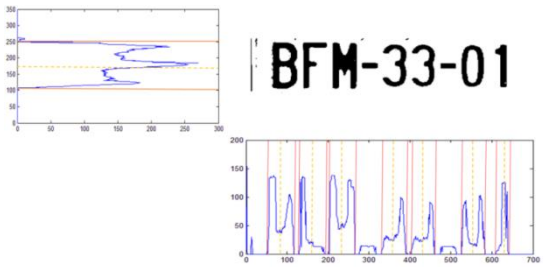

Figure 6. Horizontal and vertical projection of a binarized image.

Fig. 7 shows some examples of segmented characters used to do horizontal and vertical projections.

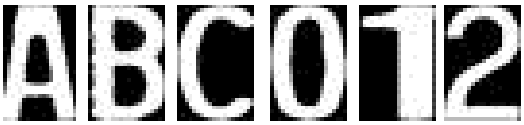

Figure 7. Examples of characters segmented using horizontal and vertical projections.

C. Modelling of the characters

The techniques of Hu moments, correlation factor and Fourier descriptors are compared because they are comparison techniques in which objective values are obtained, which identify classifications of letters and numbers that will be used in techniques such as Bayesian classifier and neural network.

Segmented characters are modeled using the Correlation factor, Hu's moment and Fourier descriptors.

1) Correlation factor: Correlation is a statistical technique that quantifies the strength of the linear relationship between two variables.

The quantification was performed using the coefficient of Pearsons correlation linear [12], whose value ranges between {-1 and 1}. Suppose we have the variables x and y, and on the other hand, O={(x_1,y_1),…,(x_m,y_m)} is the set of pixels coordinates to have a character extracted, the correlation coefficient between both variables is calculated as:

$$r = \frac{\sum_{(x,y)\in O}(x-\bar{x})(y-\bar{y})}{\sqrt{\sum_{(x,y)\in O}(x-\bar{x})^2 \sum_{(x,y)\in O}(y-\bar{y})^2}} \quad (4)$$

2) Hu's moment: Hu's moments have been employed to recognize characters in [13] and [14], to measure geometric features as ellipticities [15] or circularities [16]. In the Hu's moment, the most representative is the centralized and standardized moment, η_{pq}, obtained with:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{c+1}} \quad (5)$$

In equation (5), c and μ_{pq} is calculated as:

$$c = \frac{p+q}{2}, \qquad p+q = 2,3,\dots \quad (6)$$

$$\mu_{pq}(O) = \sum_{(x,y)\in O} (x-\bar{x})^p (y-\bar{y})^q \quad (7)$$

where $(\bar{x}, \bar{y})$ is the coordinate of the centroid of the objects, calculated as:

$$\bar{x} = \frac{1}{m}\sum_{(x,y)\in O} x \quad (8)$$

$$\bar{y} = \frac{1}{m}\sum_{(x,y)\in O} y \quad (9)$$

It is important to mention that Hu's moments are invariant to the position, rotation, and scaling.

3) Fourier Descriptors: Given $\{(x_1, y_1), \dots (x_n, y_n)\} \subset \mathbb{Z}^2$ the set of points that form the outline of a letter, each point is represented as a complex number.

$$z_n = x_n + jy_n \quad , \qquad where\ j = \sqrt{-1}. \quad (10)$$

Discrete Fourier transform of the set of points that make up the outline of the characters can be calculated as follows:

$$F(u) = \frac{1}{u\pi}\sum_{n=1}^{m} \varphi_n \left[\cos\left(\frac{2\pi u s_n}{s_m}\right) - jsin\left(\frac{2\pi u s_n}{s_m}\right)\right] \quad (11)$$

It can be written as:

$$F(u) = \frac{1}{u\pi}\sum_{n=1}^{m} \varphi_n exp\left(\frac{-2j\pi u s_n}{s_m}\right) \quad (12)$$

Finally, the Fourier descriptors are obtained when calculating the absolute values of the complex numbers:

$$f(u) = |F(u)| \quad (13)$$

where $u = 1, 2, \dots, N$ and N is the total number of descriptors to obtain.

D. Characters Recognition

For the characters recognition four techniques are used: 1) Template matching, 2) Bayesians classifier and 3) Artificial Neural Networks (ANN) and Principal Component Analysis, which will be explained in more detail below.

1) The template matching: The comparison of templates is a technique that consists of comparing the image of the character to be recognized with a series of known templates; its similarity can be measured to identify the character that contains the image to be classified. This technique has been used for the same purpose in [18] and [21]. When an image must be classified, it is compared with all the images of the templates and the maximum value of similarity is obtained:

$$\Xi_\sigma = Max(r_\sigma), \forall \sigma \in (0, 1, \dots, 9, \mathrm{A}, \mathrm{B}, \dots, \mathrm{Z}) \qquad (14)$$

With the value of Ξ_σ it is defined what symbol is the content in the image.

2) Bayesian classifier: The Bayesian classifier [19] is based on the Bayes theorem where it is assumed that the vector of characteristics is a multivariate Gaussian distribution. $C = \{k_1, \dots, k_n\}$ is the set of n class; the probability that an object A is a class k_i is denoted by $p(k_i|\boldsymbol{A})$. The Bayes theorem:

$$p(k_i|\mathbf{A}) = \frac{p(\mathbf{A}|k_i)p(k_i)}{p(\boldsymbol{A})} \qquad (15)$$

For Bayesian classification, it is chosen the class ki where $p(A|k_i)p\ (k_i)$ is larger. This way the observed object **A** is assigned to the class ki. We assume that the probability of the feature vector of object A is of class *kj* $(p(\mathbf{A}|k_i))$, has a Gaussian Distribution with mean μ and with covariance matrix **Ω** defined as follows:

$$P(A|k_i) = \frac{1}{\Delta} exp\left[-\frac{1}{2}\left(A - \mu_j\right)^T \Omega_j^{-1}\left(A - \mu_j\right)\right]$$

Where $\Delta = (2\pi)^{m/2}\left(\det \Omega_j\right)^{1/2}$ and m is the dimension of the feature vectors.

3) Artificial Neural Network: It is a mathematical model that is inspired by the way biological neurons work. The ANN [20] is applied in the learning of tasks where each instance is described by a set of values that represent its Special features and where the function objective **f (A)** can take any value from a finite set **V**. In the present work, there are implemented two ANN, one dedicated to the classification of numbers and another for letters. The ANN deployed is of type Backpropagation, activated by the sigmoid function.

4) Principal Component Analysis: The Principal Components Analysis (PCA) is a technique in which a set of correlated variables are transformed to another set of not correlated variables that is a linear combination from the original variables in which most of these variables can be removed with minimal loss of the original information; this characteristic allows the PCA to be employed to reduce the dimensionality of a large set of data losing the least of information.

III. EXPERIMENTS

To perform the experimentations phase it was employed a database of 70 images of plates from the 31 States of the Mexican Republic, except Mexico City because these entity license plates are different (contain 6 characters instead of 7). This database was chosen because it was considered the sampling by convenience. The images were captured at a distance of 1.5 to 3 meters between the camera and the plate, seeking a uniform lighting and subsequently normalized to 700 × 350 pixels.

Experimentation was made in two phases:

- Training to recognize letters and numbers
- License plate recognition.

A. Training torecognizer letters and numbers

To test the recognizers, 738 characters, 374 letters, and 364 numbers were used. In Table I we showed the number of characters used in each case. That number is not equal for all symbols since they do not appear with the same frequency in the license plate.

Three types of characters recognizers were implemented: 1) coefficient of correlation 2) Bayesian classifier and 3) Artificial Neural Network (ANN). These last 2 networks were implemented, one for numbers and another for letters.

TABLE I. NUMBER OF EXAMPLES USED DURING THE TRAINING.

A	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*J*	*K*	*L*
14	9	11	17	16	18	26	21	20	13	14
M	N	R	P	S	T	U	V	W	X	Y
13	14	20	17	12	18	18	21	13	15	18
Z	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
16	27	27	40	32	41	50	43	34	33	37

In the case of the Bayesian classifier and ANN, a vector of 8-dimension characteristics was used at first, formed by 7 Hu's moment and correlation Factor; the results to classify the numbers and letters are shown in Table II. In the present experiment, the networks are two layers with the following structure: {8,10} for numbers and {8,23} for letters.

TABLE II. RESULTS OF CLASSIFICATION WITH 7 HU'S MOMENT AND THE CORRELTION FACTOR

	Bayesian classifier			
Character	**Hits**	**%**	**Hits**	**%**
Numbers	246/364	67.58	333/364	91.48
Letters	294/374	78.60	273/374	72.99

 ISBN: 978-1-61208-613-2

Subsequently, vectors were added with characteristics 30 and 60 Fourier descriptors, forming vectors of dimension 38: 7 Hu's moment, 30 Fourier descriptors and correlation factor, and dimension 68: correlation factor, 7 Hu's moment and 60 Fourier descriptors. Results obtained in the tests with the Bayesian classifier can be observed in Table III.

In this experiment, neural networks were also implemented for the 38 and 68 features, establishing several elements per layer experimentally. For example, the structure for network with 38 features is: letters = {38, 76, 23} and numbers = {38, 38, 10}. Both networks correctly classified 100% of the examples.

TABLE III. BAYESIAN CLASSIFIER RESULTS WITH 38 AND 68 FEATURES.

	Bayesian classifier			
Character	**38 hits**	**%**	**68 hits**	**%**
Numbers	325/364	89.40	331/364	91.50
Letters	360/374	96.4	369/374	98.7

Finally, the Principal Components Analysis was observed to reduce the dimension of the vector of characteristics in the following way:

1) 38 features vector: The number of vectors decreased to 9 components, while the letters vectors went from 38 to 8 items. Neural networks implemented in this case have the following composition: numbers: {9, 27, 36, 10}, Letters: {8, 16, 32, 23}.

2) 68 features vector: The number of vectors decreased to 13 components, while the letters vectors went to only 11. Neural networks implemented in this case have the following composition: Numbers: {13, 26, 13, 10}, Letters: {11, 22, 46, 23}.

B. Recognition of Vehicle`s registration.

Table IV shows the number of images of plates used by State.

TABLE IV. EXAMPLES NUMBER OF REGISTRATIONS BY STATE.

Aguascalientes	Baja California Norte	Baja California Sur	Campeche	Chihuahua
2	1	0	2	2
Colima	Coahuila	Chiapas	Durango	Estado de México
1	2	2	1	3
Guerrero	Guanajuato	Hidalgo	Jalisco	Michoacán
3	4	3	4	4
Morelos	Nuevo León	Nayarit	Oaxaca	Puebla
2	2	3	2	2
Querétaro	Quintana Roo	San Luis Potosí	Sinaloa	Sonora
4	1	4	3	1
Tabasco	Tamaulipas	Tlaxcala	Veracruz	Yucatán
2	2	1	2	2
Zacatecas	Total			
3	70			

The results obtained in the test what the recognizers developed are shown in Figure 8.

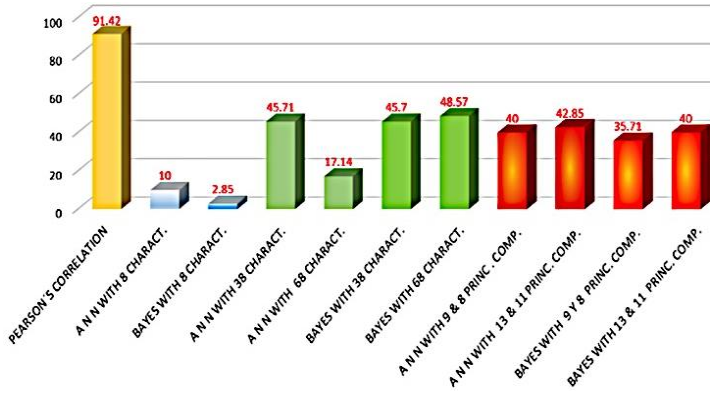

Figure 8. Results of classification plates to different three organizers implemented.

To consider recognition of the registration as a success we use the following criteria: A registration is successfully recognized if all their characters are classified correctly. Otherwise, those registrations are not recognized successfully.

IV. DISCUSSION

In the development of this proposal, there were implemented 3 types of recognizers: 1) correlation factor, 2) Bayesian classifier and 3) Artificial neuronal network. Out of these techniques, the correlation Factor had the best performance as it can be seen in Fig. 8. The main problem with this technique is comparing the examples of plates presented in Table IV with a properly chosen template. This is due, on one hand, to the segmented characters that may have an excess of noise, and on the other hand, to the size and shape of the character.

Using the Bayesian classifier with only 8 features we got a percentage of hits of 2.85%. However, when it increased the number of features with Fourier descriptors, the percentage increased to 45.7% with 38 features and 48.57% with 68 characteristics. When the Principal Components was incorporated, the percentage decreased by 8%. This shows that the cumulative variance cannot be greater than 95%.

For neural networks, with 38 features we obtained 47.50% of hits and 17.14% with 68 features. When using 8 and 9 principal components with 38 features, performance decreased only by 5%. On the other hand, when using 13 and 11 principal components with 68 features, the number of hits increased to 43.85%. This shows that increasing the number of elements in character modeling is not always the best alternative.

The classification mistakes in the Bayesian classifier recognizer is presented in the following form: (a) Number 9 is classified as 6 and 4, 1 is recognized as 7 and 8 is

 ISBN: 978-1-61208-613-2

classified as 0. (b) Letters: J and T are classified as L and H, respectively, and letters X and M are recognized as W.

The classification mistakes in artificial neural networks are: (a) Number 9 is recognized as 6 and vice versa, 5 as 3 and in some cases 4 as 9. (b) Letter R is classified as K; L is recognized as J; M is rated as W and vice versa; V is recognized as A.

In the two previous classifiers, an error that occurs in both is that they cannot recognize characters that have a very similar structure; some examples are 5 and 8 and H and K, or characters that look like their rotated version such as M and W, 6 and 9, and V and A.

V. CONCLUSIONS

In this paper, we proposed an algorithm for recognition of vehicle registration plates of Mexico, which consider different texture patterns and colors in the background of plates. This is an improvement with respect to other works, since most consider plates formed with black characters on white background.

In the case of the vehicle registration plates of Mexico, they have different texture patterns and colors in the background; so, when we applied the traditional algorithms, their performance was considerably affected. This was mainly due to the segmentation of characters; they are usually extracted with part of the texture of the background, getting incorrect characters.

The results showed that the proposed algorithm is robust because in tests recognition using the Pearson correlation coefficient we obtained a 91.42% of recognized characters. In this case, we used techniques of template matching, which showed that characters segmentation was adequate.

This proposed algorithm used the Mexican official standard NOM-001-SCT-2-2000, which specifies the dimensions of the plate, and the size and location of the characters; this information provided the phase of segmentation of characters.

By experimentation, it was obtained a threshold of pixel intensity for each character that allows us to separate them from the background characters with more precision.

Although the recognition was minimal, it should be considered that the misclassification of a single character implies that the plate is not acknowledge successfully, therefore, a finer threshold to decide when a plaque is or is not recognized must be established.

Using the characterization of the correlation coefficient, Hu's moments and Fourier descriptors allowed to recognize the characters successfully. So, even though results obtained in the phase of license plate recognition were low, good results were achieved in an individual way.

For some alphanumeric characters of the plates, such as the M and W, 6 and 9, A and V, based on the characteristic that the moments of Hu are invariant, the recognition was wrong. This was because, in an invariant process, the characters are described through a set of quantifiable features (very similar in the previous characters), that are insensitive to any type of deformation. For the rotation characters in the Hu's moments is necessary to apply mechanisms to reduce the number of misrecognized characters, for example, Chain Codes. For these, we chose a starting point and travelled the border in a clockwise direction indicating the direction the border is following, thus having a qualitative and quantitive recognition.

This study is original with respect to Automatic Number Plate Recognition (ANPR) literature and addresses harder use-case than those commonly evaluated. We believe the performances of the proposed algorithm should be compared to open-source ANPR engines addressing textured immatriculation plates management.

The dataset we used in this study is small: 2 plates per Mexican state, resulting in 9 to 26 examples for each letter, which is rather low to train and evaluate robust systems. This is the reason why it is necessary to increase the size of the database to improve the results of the proposed algorithm.

REFERENCES

[1] Gazcón, N. F. (2012). Automatic vehicle identification for Argentinean license plates using intelligent template matching. *Pattern Recognition Letters , 33* (9), 1066--1074.

[2] Siddique, N. A. (2012). Development of an automatic vehicle license plate detection and recognition system for Bangladesh. *Electronics & Vision (ICIEV), 2012 International Conference on Informatics* , 688--693.

[3] Y. Wang, J. C. (2015). License Plate Recognition Based on SIFT Feature. *Optik - International Journal for Light and Electron Optics*.

[4] M.A. Massoud, M. S. (2013). Automated new license plate recognition in Egypt. *Alexandria Engineering Journal*.

[5] Nasipuri, S. S. (2014). iLPR: an Indian license plate recognition system. *Multimedia Tools and Applications*.

[6] Cheng, Y. a. (2004). Car license plate recognition based on the combination of principal components analysis and radial basis function networks. *7th International Conference on Signal Processing , 2*, 1455--1458.

[7] Al Faqheri, W. a. (2009). A real-time Malaysian automatic license plate recognition (M-ALPR) using hybrid fuzzy. *IJCSNS International Journal of Computer Science and Network Security , 9* (2), 333--340.

[8] Otsu, N. (1979). A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man and Cybernetics* , 62-66.

[9] Erick Cuevas, D. Z. (2010). Procesamiento digital de imagenes con MATLAB y Simulink. México: Alfaomega Ra-Ma.

[10] Duan, T. D. (2005). Building an automatic vehicle license plate recognition system. *Proc. Int. Conf. Comput. Sci. RIVF* , 59--63.

[11] Qin, Z. a. (2006). Method of license plate location based on corner feature. *6th World Congress on Intelligent Control and Automation , 2*, 8645--8649.

[12] A. Lind Douglas, A. W. (2012). *Estadística Aplicada a los Negocios y la Economía.* Mexico D. F.: McGraw Hill.

[13] Wong, W.-H. a.-C.-M. (1995). Generation of moment invariants and their uses for character recognition. *Pattern Recognition Letters , 16*, 115--123.

[14] Gómez González, S. a. (2011). Desarrollo de un sistema prototipo de reconocimiento de digitos usando momentos invariantes. Universidad Tecnológica de Pereira.

 ISBN: 978-1-61208-613-2

[15] Zunic, D. a. (2014). Shape ellipticity from Hu moment invariants. *Applied Mathematics and Computation , 226*, 406--414.

[16] Zunic, J. a. (2010). A Hu moment invariant as a shape circularity measure. *Pattern Recognition , 43* (1), 47--57.

[17] Steven L. Eddins Rafael C. Gonzalez, R. E. (2004.). *Digital Image Processing Using.* Pearson Prentice Hall.

[18] Du, S. a. (2013). Automatic license plate recognition (ALPR): A state-of-the-art review. *IEEE Transactions on circuits and systems for video technology , 23* (2), 311--325.

[19] Tom, M. M. (1997). *Machine Learning.* Ithaca, N. Y.: McGraw-Hill.

[20] T. Hagan Martín, B. D. (2014). *Neural Network Design.* **ISBN-10:** 0971732116

[21] Hu, M.-K. (1962). Visual Pattern Recognition by Moment Invariants. *IRE Transactions on Information Theory* , 179-1z

 ISBN: 978-1-61208-613-2

Deep Learning: A Tool for Computational Nuclear Physics

Gianina Alina Negoita*†, Glenn R. Luecke‡, James P. Vary§, Pieter Maris§, Andrey M. Shirokov¶‖,
Ik Jae Shin**, Youngman Kim**, Esmond G. Ng†† and Chao Yang††

*Department of Computer Science, Iowa State University, Ames, Iowa, USA
Email: alina@iastate.edu
†Horia Hulubei National Institute for Physics and Nuclear Engineering, Bucharest-Magurele 76900, Romania
‡Department of Mathematics, Iowa State University, Ames, Iowa, USA
Email: grl@iastate.edu
§Department of Physics and Astronomy, Iowa State University, Ames, Iowa, USA
Email: jvary@iastate.edu, pmaris@iastate.edu
¶Skobeltsyn Institute of Nuclear Physics, Moscow State University, Moscow 119991, Russia
Email: shirokov@nucl-th.sinp.msu.ru
‖Department of Physics, Pacific National University, Khabarovsk 680035, Russia
**Rare Isotope Science Project, Institute for Basic Science, Daejeon 34047, Korea
Email: geniean@ibs.re.kr, ykim@ibs.re.kr
††Lawrence Berkeley National Laboratory, Berkeley, California, USA
Email: egng@lbl.gov, cyang@lbl.gov

***Abstract*—In recent years, several successful applications of the Artificial Neural Networks (ANNs) have emerged in nuclear physics and high-energy physics, as well as in biology, chemistry, meteorology, and other fields of science. A major goal of nuclear theory is to predict nuclear structure and nuclear reactions from the underlying theory of the strong interactions, Quantum Chromodynamics (QCD). With access to powerful High Performance Computing (HPC) systems, several ab initio approaches, such as the No-Core Shell Model (NCSM), have been developed to calculate the properties of atomic nuclei. However, to accurately solve for the properties of atomic nuclei, one faces immense theoretical and computational challenges. The present study proposes a *feed-forward* ANN method for predicting the properties of atomic nuclei like ground state energy and ground state point proton root-mean-square (rms) radius based on NCSM results in computationally accessible basis spaces. The designed ANNs are sufficient to produce results for these two very different observables in ^{6}Li from the ab initio NCSM results in small basis spaces that satisfy the theoretical physics condition: independence of basis space parameters in the limit of extremely large matrices. We also provide comparisons of the results from ANNs with established methods of estimating the results in the infinite matrix limit.**

Keywords–Nuclear structure of ^{6}Li; ab initio no-core shell model; ground state energy; point proton root-mean-square radius; artificial neural network.

I. INTRODUCTION

Nuclei are complicated quantum many-body systems, whose inter-nucleon interactions are not known precisely. The goal of ab initio nuclear theory is to accurately describe nuclei from the first principles as systems of nucleons that interact by fundamental interactions. With sufficiently precise many-body tools, we learn important features of these interactions, such as the fact that three-nucleon (NNN) interactions are critical for understanding the anomalous long lifetime of ^{14}C [1]. With access to powerful High Performance Computing (HPC) systems, several ab initio approaches have been developed to study nuclear structure and reactions, such as the No-Core Shell Model (NCSM) [2], the Green's Function Monte Carlo (GFMC) [3], the Coupled-Cluster Theory (CC) [4], the Hyperspherical expansion method [5], the Nuclear Lattice Effective Field Theory [6][7], the No-Core Shell Model with Continuum [2] and the NCSM-SS-HORSE approach [8]. These approaches have proven to be successful in reproducing the experimental nuclear spectra for a small fraction of the estimated 7000 nuclei produced in nature.

The ab initio theory may employ a high-quality realistic nucleon-nucleon (NN) interaction, which gives an accurate description of NN scattering data and predictions for binding energies, spectra and other observables in light nuclei. Daejeon16 is a NN interaction [9] based on Chiral Effective Field Theory (χEFT), a promising theoretical approach to obtain a quantitative description of the nuclear force from the first principles [10]. This interaction has been designed to describe light nuclei without explicit use of NNN interactions, which require a significant increase of computational resources. It has also been shown that this interaction provides good convergence of many-body ab initio NCSM calculations [9].

Properties of ^{6}Li and other nuclei, such as ^{3}H, ^{3}He, ^{4}He, ^{6}He, ^{8}He, ^{10}B, ^{12}C and ^{16}O, were investigated using the ab initio NCSM approach with the Daejeon16 NN interaction and compared with JISP16 [11] results. The results showed that Daejeon16 provides both improved convergence and better agreement with data than JISP16. These calculations were performed with the code MFDn [12]–[14], a hybrid MPI/OpenMP code for ab initio nuclear structure calculations. However, one faces major challenges to approach convergence since, as the basis space increases, the demands on computational resources grow very rapidly.

The present work proposes a *feed-forward* Artificial Neural Network (ANN) method as a different approach for obtaining the properties of atomic nuclei such as the ground state (gs) energy and the ground state (gs) point proton root-mean-square (rms) radius based on results from readily-solved basis spaces. *Feed-forward* ANNs can be viewed as universal

 ISBN: 978-1-61208-613-2

non-linear function approximators [15]. Moreover, ANNs can find solution when algorithmic methods are computationally intensive or do not exist. For this reason, ANNs are considered a more powerful modeling method for mapping complex non-linear input-output problems. The output values of ANNs are obtained by simulating the human learning process from the set of learning examples of the input-output association provided to the network. Additional information about ANNs can be found in [16][17].

Although the gs energy and the gs point proton rms radius are ultimately determined by complicated many-body interactions between the nucleons, the variation of the NCSM calculation results appears to be smooth with respect to the two basis space parameters, $\hbar\Omega$ and $N_{\max}$, where $\hbar\Omega$ is the harmonic oscillator (HO) energy and $N_{\max}$ is the basis truncation parameter. In practice, these calculations are limited and one can not calculate the gs energy or the gs point proton rms radius for very large $N_{\max}$. To obtain the gs energy and the gs point proton rms radius as close as possible to the exact results, the results are extrapolated to the infinite model space. However, it is difficult to construct a simple function with a few parameters to model this type of variation and extrapolate the results to the infinite matrix limit. The advantage of ANN is that it does not need an explicit analytical expression to model the variation of the gs energy or the gs point proton rms radius with respect to $\hbar\Omega$ and $N_{\max}$. The *feed-forward* ANN method is very useful to find the converged result at very large $N_{\max}$.

In recent years, ANNs have been used in many areas of nuclear physics and high-energy physics. In nuclear physics, ANN models have been developed for constructing a model for the nuclear charge radii [18], determination of one and two proton separation energies [19], developing nuclear mass systematics [20], identification of impact parameter in heavy-ion collisions [21]–[23], estimating beta decay half-lives [24] and obtaining potential energy curves [25]. In high-energy physics, ANNs are used routinely in experiments for both online triggers and offline data analysis due to an increased complexity of the data and the physics processes investigated. Both the DIRAC [26] and the H1 [27] experiments used ANNs for triggers. For offline data analysis, ANNs were used or tested for a variety of tasks, such as track and vertex reconstruction (DELPHI experiment [28]), particle identification and discrimination (decay of the Z^0 boson [29]), calorimeter energy estimation and jet tagging. Tevatron experiments used ANNs for the direct measurement of the top quark mass [30] or leptoquark searches [31]. In terms of types of ANNs, the vast majority of applications in nuclear physics and high-energy physics were based on *feed-forward* ANNs, other types of ANNs remaining almost unexplored. An exception is the DELPHI experiment, which used a recurrent ANN for tracking reconstruction [28].

This research presents results for two very different physical observables for ^{6}Li, gs energy and gs point proton rms radius, produced with the *feed-forward* ANN method. Theoretical data for ^{6}Li are available from the ab initio NCSM calculations with the MFDn code using the Daejeon16 NN interaction and HO basis spaces up through the cutoff $N_{\max} = 18$. This cutoff is defined for ^{6}Li as the maximum total HO quanta allowed in the Slater determinants forming the basis space less 2 quanta. The dimension of the resulting many-body Hamiltonian matrix is about 2.8 billion at this cutoff. We return to discussing the many-body HO basis shortly. However, for the training stage of ANN, data up through $N_{\max} = 10$ was used, where the Hamiltonian matrix dimension for ^{6}Li is only about 9.7 million. Comparisons of the results from *feed-forward* ANNs with established methods of estimating the results in the infinite matrix limit are also provided. The paper is organized as follows: In Section II, short introductions to the ab initio NCSM method and ANN's formalism are given. In Section III, our ANN's architecture is presented. Section IV presents the results and discussions of this work. Section V contains our conclusion and future work.

II. Theoretical Framework

The NCSM is an ab initio approach to the nuclear many-body problem for light nuclei, which solves for the properties of nuclei for an arbitrary NN interaction, preserving all the symmetries. Naturally, the results obtained with this method are limited to the largest computationally feasible basis space. We will show that the ANN method is useful to make predictions at ultra-large basis spaces using available data from NCSM calculations at smaller basis spaces. More discussions on these two methods are presented in each subsection.

A. Ab initio NCSM Method

In the NCSM method, the neutrons and protons (separate species of nucleons) interact independently with each other. The Hamiltonian of A nucleons contains kinetic energy ($T_{\rm rel}$) and interaction (V) terms

$$\begin{aligned} H_A &= T_{\rm rel} + V \\ &= \frac{1}{A}\sum_{i<j}\frac{(\vec{p}_i - \vec{p}_j)^2}{2m} + \sum_{i<j}^{A} V_{ij} + \sum_{i<j<k}^{A} V_{ijk} + \ \ldots, \end{aligned} \tag{1}$$

where m is the nucleon mass, $\vec{p}_i$ is the momentum of the i-th nucleon, V_{ij} is the NN interaction including the Coulomb interaction between protons and V_{ijk} is the NNN interaction. Higher-body interactions are also allowed and signified by the three dots. The HO center-of-mass (CM) Hamiltonian with a Lagrange multiplier is added to the Hamiltonian above to force the many-body eigenstates to factorize into a CM component times an intrinsic component as in [32]. This way, the spurious CM excited states are pushed up above the physically relevant states, which have the lowest eigenstate of the HO for CM motion.

With the nuclear Hamiltonian specified above in (1), the NCSM solves the A-body Schrödinger equation using a matrix formulation

$$H_A\Psi_A(\vec{r}_1, \vec{r}_2, \ldots, \vec{r}_A) = E\Psi_A(\vec{r}_1, \vec{r}_2, \ldots, \vec{r}_A), \tag{2}$$

where the A-body wave function is given by a linear combination of Slater determinants ϕ_i

$$\Psi_A(\vec{r}_1, \vec{r}_2, \ldots, \vec{r}_A) = \sum_{i=0}^{k} c_i\phi_i(\vec{r}_1, \vec{r}_2, \ldots, \vec{r}_A), \tag{3}$$

and where k is the number of many-body basis states, configurations, in the system. To obtain the exact A-body wave function one has to consider infinite number of configurations, $k = \infty$. However, in practice, the sum is limited to a finite number of configurations determined by $N_{\max}$. The Slater determinant ϕ_i is the antisymmetrized product of single particle wave functions $\phi_\alpha(\vec{r})$, where α stands for the quantum

 ISBN: 978-1-61208-613-2

numbers of a single particle state. A common choice for the single particle wave functions is the HO basis functions. The matrix elements of the Hamiltonian in the many-body HO basis is given by $H_{ij} = \langle\phi_i|\hat{H}|\phi_j\rangle$. For these large and sparse Hamiltonian matrices, the Lanczos method is one possible choice to find the extreme eigenvalues [33].

To be more specific, our limited many-body HO basis is characterized by two basis space parameters: $\hbar\Omega$ and $N_{\max}$, where $\hbar\Omega$ is the HO energy and $N_{\max}$ is the basis truncation parameter. In this approach, all possible configurations with $N_{\max}$ excitations above the unperturbed gs (the HO configuration with the minimum HO energy defined to be the $N_{\max} = 0$ configuration) are considered. Even values of $N_{\max}$ correspond to states with the same parity as the unperturbed gs and are called the "natural" parity states, while odd values of $N_{\max}$ correspond to states with "unnatural" parity.

Due to the strong short-range correlations of nucleons in a nucleus, a large basis space, or model space, one that is often not feasible, is required to achieve convergence. To obtain the gs energy and other observables as close as possible to the exact results one has to choose the largest feasible basis spaces. Next, if numerical convergence is not achieved, which is often the case, the results are extrapolated to the infinite model space. To take the infinite matrix limit, several extrapolation methods have been developed (see, for example, [34]).

B. Artificial Neural Networks

ANNs are powerful tools that can be used for function approximation, classification and pattern recognition, such as finding clusters or regularities in the data. The goal of ANNs is to find a solution efficiently when algorithmic methods are computationally intensive or do not exist. An important advantage of ANNs is the ability to detect complex non-linear input-output relationships. For this reason, ANNs can be viewed as universal non-linear function approximators [15]. Employing ANNs for mapping complex non-linear input-output problems offers a significant advantage over conventional techniques, such as regression techniques, because ANNs do not require explicit mathematical functions.

ANNs are defined as computer algorithms that mimic the human brain, being inspired by biological neural systems. Similar to the human brain, ANNs can perform complex tasks, such as learning, memorization and generalization. They are capable of learning from experience, storing knowledge and then applying this knowledge to make predictions.

A biological neuron has a cell body, a nucleus, dendrites and an axon. Dendrites act as inputs, the axon propagates the signal and the interaction between neurons takes place at synapses. Each synapse has an associated weight. When a neuron 'fires', it sends an output through the axon and the synapse to another neuron. Each neuron then collects all the inputs coming from linked neurons and produces an output.

The artificial neuron (AN) is a model of the biological neuron. Figure 1 shows a representation of an AN. Similarly, the AN receives a set of input signals $(x_1, x_2, ..., x_n)$ from an external source or from another AN. A weight w_i $(i = 1, ..., n)$ is associated with each input signal x_i $(i = 1, ..., n)$. Additionally, each AN that is not in the input layer has another input signal called the *bias* with value 1 and its associated weight b. The AN collects all the input signals and calculates a *net* signal as the weighted sum of all input signals as

$$net = \sum_{i=1}^{n+1} w_i x_i, \tag{4}$$

where $x_{n+1} = 1$ and $w_{n+1} = b$.

Next, the AN calculates and transmits an output signal, y. The output signal is calculated using a function called an *activation* or *transfer* function, which depends on the value of the *net* signal, $y = f(net)$.

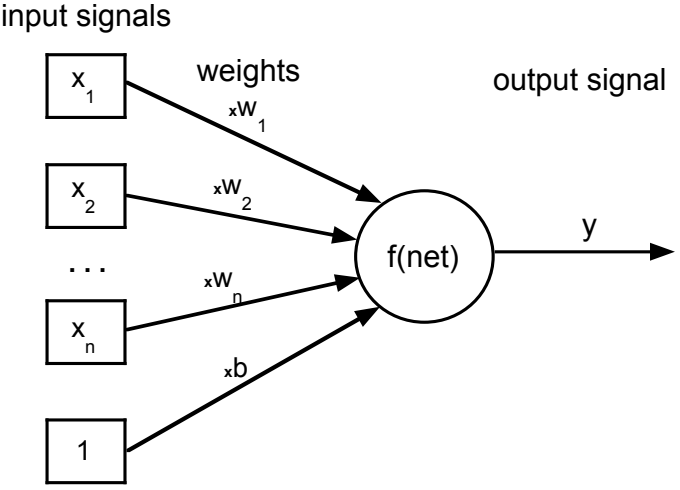

Figure 1. An artificial neuron.

ANNs consist of a number of highly interconnected ANs which are processing units. One simple way to organize ANs is in layers, which gives a class of ANN called multi-layer ANN. ANNs are composed of an input layer, one or more hidden layers and an output layer. The neurons in the input layer receive the data from outside and transmit the data via weighted connections to the neurons in the hidden layer, which, in turn, transmit the data to the next layer. Each layer transmits the data to the next layer. Finally, the neurons in the output layer give the results. The type of ANN, which propagates the input through all the layers and has no *feed-back* loops is called a *feed-forward* multi-layer ANN. For simplicity, throughout this paper we adopt and work with a *feed-forward* ANN. For other types of ANN, see [16][17].

Figure 2 shows an example of a *feed-forward* three-layer ANN. It contains one input layer, one hidden layer and one output layer. The input layer has n ANs, the hidden layer has m ANs and the output layer has p ANs. The connections between the neurons are weighted as follows: v_{ji} are the weights between the input layer and the hidden layer, and w_{kj} are the weights between the hidden layer and the output layer, where $(i = 1, ..., n)$, $(j = 1, ..., m)$ and $(k = 1, ..., p)$. In this example, the input layer has no *activation* function, the hidden layer has *activation* function f and the output layer has *activation* function g. It is also possible to have a different *activation* function for each individual neuron.

The *activation* function in the hidden layer, f, is different from the *activation* function in the output layer, g. For function approximation, a common choice for the *activation* function for the neurons in the hidden layer is a *sigmoid* or *sigmoid*–like function, while the neurons in the output layer have a *linear*

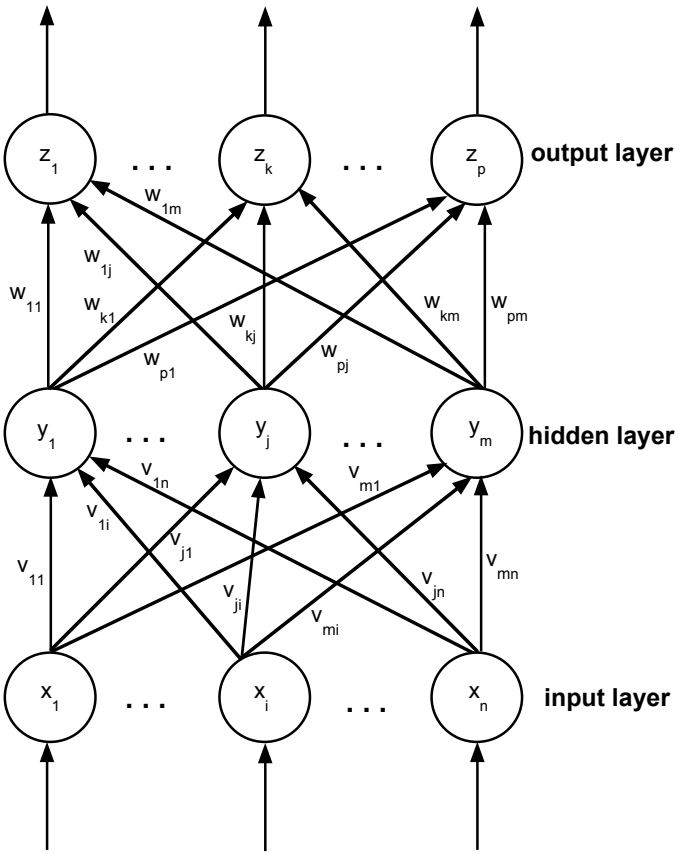

Figure 2. A three-layer ANN.

function:

$$f(x) = \frac{1}{1 + e^{-ax}}, \tag{5}$$

where a is the slope parameter of the *sigmoid* function and

$$g(x) = x. \tag{6}$$

The neurons with *non-linear activation* functions allow the ANN to learn *non-linear* and *linear* relationships between input and output vectors. Therefore, sufficient neurons should be used in the hidden layer in order to get a good function approximation.

In the example shown in Figure 2 and with the notations mentioned above, the network propagates the external signal through the layers producing the output signal z_k at neuron k in the output layer

$$\begin{aligned} z_k = g(net_{z_k}) &= g(\sum_{j=1}^{m+1} w_{kj} f(net_{y_j})) \\ &= g(\sum_{j=1}^{m+1} w_{kj} f(\sum_{i=1}^{n+1} v_{ji} x_i)). \end{aligned} \tag{7}$$

The use of an ANN is a two-step process, training and testing stages. In the training stage, the ANN adjusts its weights until an acceptable error level between desired and predicted outputs is obtained. The difference between desired and predicted outputs is measured by the error function, also called the performance function. A common choice for the error function is *mean square error* (MSE).

There are multiple training algorithms based on various implementations of the *back-propagation* algorithm [35], an efficient method for computing the gradient of error functions. These algorithms compute the *net* signals and outputs of each neuron in the network every time the weights are adjusted as in (7), the operation being called the *forward pass* operation. Next, in the *backward pass* operation, the errors for each neuron in the network are computed and the weights of the network are updated as a function of the errors until the stopping criterion is satisfied. In the testing stage, the trained ANN is tested over new data that was not used in the training process. The predicted output is calculated using (7).

One of the known problems for ANN is overfitting: the error on the training set is within the acceptable limits, but when new data is presented to the network the error is large. In this case, ANN has memorized the training examples, but it has not learned to generalize to new data. This problem can be prevented using several techniques, such as early stopping, regularization, weight decay, *hold-out* method, *m-fold cross-validation* and others.

Early stopping is widely used. In this technique the available data is divided into three subsets: the training set, the validation set and the test set. The training set is used for computing the gradient and updating the network weights and biases. The error on the validation set is monitored during the training process. When the validation error increases for a specified number of iterations, the training is stopped, and the weights and biases at the minimum of the validation error are returned. The test set error is not used during training, but it is used as a further check that the network generalizes well and to compare different ANN models.

Regularization modifies the performance function by adding a term that consists of the mean of the sum of squares of the network weights and biases. However, the problem with regularization is that it is difficult to determine the optimum value for the performance ratio parameter. It is desirable to determine the optimal regularization parameters automatically. One approach to this process is the Bayesian regularization of David MacKay [36]. The Bayesian regularization algorithm updates the weight and bias values according to *Levenberg-Marquardt* [35][37] optimization. It minimizes a linear combination of squared errors and weights and it also modifies the regularization parameters of the linear combination to generate a network that generalizes well. See [36][38] for more detailed discussions of Bayesian regularization.

For further and general background on the ANN and how to prevent overfitting and improve generalization refer to [16][17].

III. ANN DESIGN

The topological structure of ANNs used in this study is presented in Figure 3. The designed ANNs contain one input layer with two neurons, one hidden layer with eight neurons and one output layer with one neuron. The inputs were the basis space parameters: the HO energy, $\hbar\Omega$, and the basis truncation parameter, $N_{\max}$, described in Section II. The desired outputs were the gs energy and the gs point proton rms radius of ^{6}Li. An ANN was designed for each desired output: one ANN for gs energy and another ANN for gs point proton rms radius. The optimum number of neurons in the hidden layer was obtained according to a trial and error process.

 ISBN: 978-1-61208-613-2

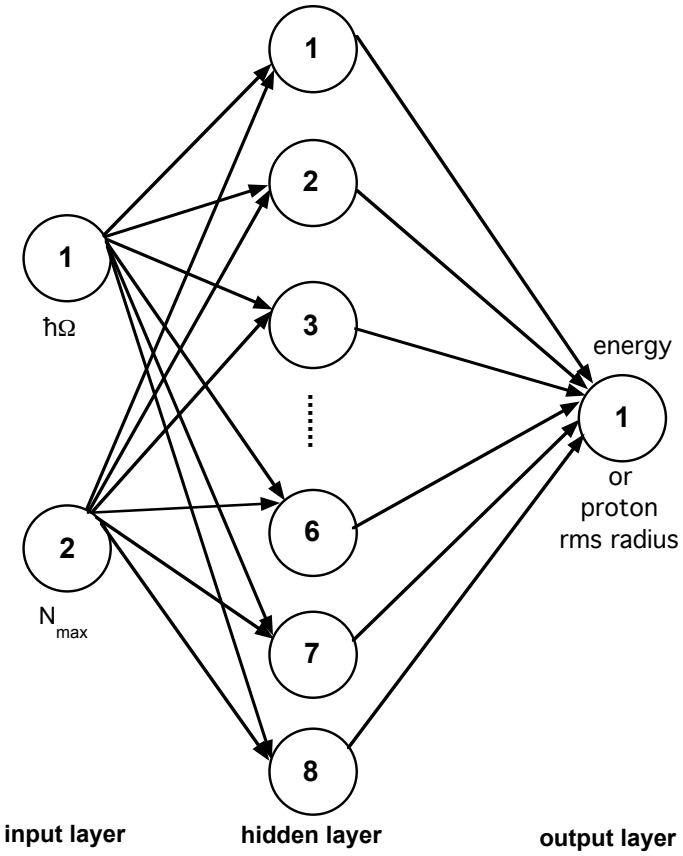

Figure 3. Topological structure of the designed ANN.

The *activation* function employed for the hidden layer was a widely-used form, the *hyperbolic tangent sigmoid* function

$$f(x) = tansig(x) = \frac{2}{(1+e^{-2x})} - 1, \qquad (8)$$

where x is the input value of the hidden neuron and $f(x)$ is the output of the hidden neuron. $tansig$ is mathematically equivalent to the *hyperbolic tangent function*, $tanh$, but it improves network functionality because it runs faster than $tanh$. It has been proven that one hidden layer and *sigmoid*-like *activation* function in this layer are sufficient to approximate any continuous real function, given sufficient number of neurons in the hidden layer [39].

MATLAB software v9.2.0 (R2017a) with *Neural Network Toolbox* was used for the implementation of this work. As mentioned before in Section I, the data set for ^{6}Li was taken from the ab initio NCSM calculations with the MFDn code using the Daejeon16 NN interaction [9] and basis spaces up through $N_{\max} = 18$. However, only the data with even $N_{\max}$ values corresponding to "natural" parity states and up through $N_{\max} = 10$ was used for the training stage of the ANN. The training data was limited to $N_{\max} = 10$ and below since future applications to heavier nuclei will likely not have data at higher $N_{\max}$ values due to exponential increase in the matrix dimension. This $N_{\max} \leq 10$ data set was randomly divided into two separate sets using the *dividerand* function in MATLAB: 85% for the training set and 15% for the testing set. A *back-propagation* algorithm with Bayesian regularization with MSE performance function was used for ANN training. Bayesian regularization does not require a validation data set.

For function approximation, Bayesian regularization provides better generalization performance than early stopping in most cases, but it takes longer to converge. The performance improvement is more noticeable when the data set is small because Bayesian regularization does not require a validation data set, leaving more data for training. In MATLAB, Bayesian regularization has been implemented in the function *trainbr*. When using *trainbr*, it is important to train the network until it reaches convergence. In this study, the training process is stopped if: (1) it reaches the maximum number of iterations, 1000; (2) the performance has an acceptable level; (3) the estimation error is below the target; or (4) the Levenberg-Marquardt adjustment parameter μ becomes larger than 10^{10}. A good typical indication for convergence is when the maximum value of μ has been reached. During training, one can choose to show the Neural Network Training tool (nntraintool) GUI in MATLAB to monitor the training progress. Figure 4 illustrates a training example as it appears in nntraintool.

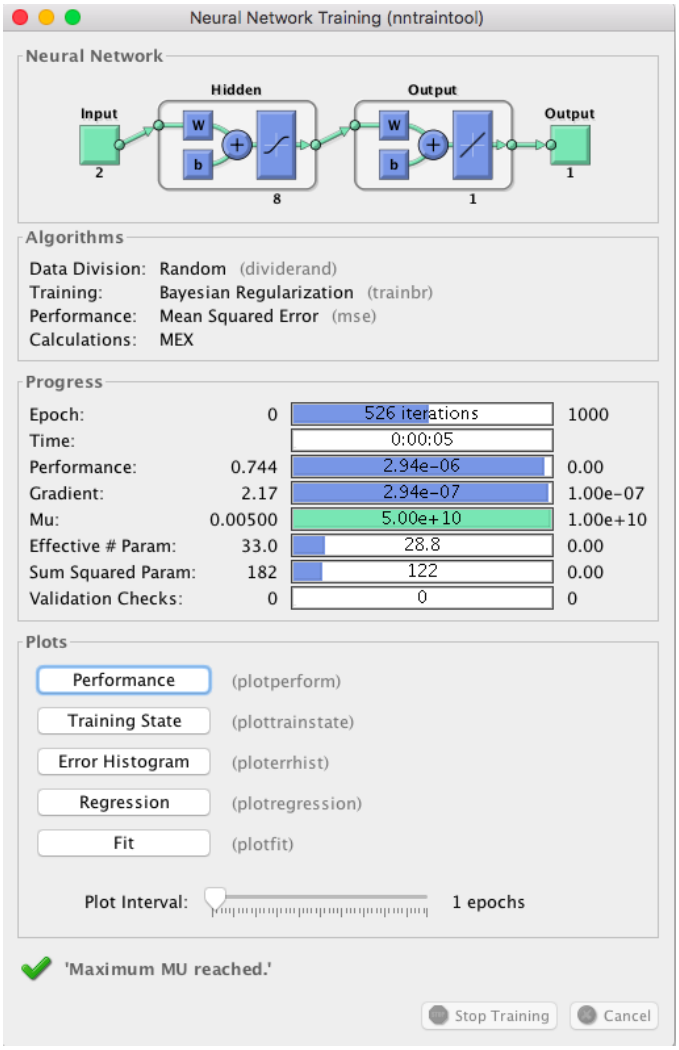

Figure 4. Neural Network Training tool (nntraintool) in MATLAB.

Note the ANN architecture view and the training stopping parameters with their ranges.

 ISBN: 978-1-61208-613-2

IV. Results and Discussions

Every ANN creation and initialization function starts with different initial conditions, such as initial weights and biases, and different division of the training, validation, and test data sets. These different initial conditions can lead to very different solutions for the same problem. Moreover, it is also possible to fail in obtaining realistic solutions with ANNs for certain initial conditions. For this reason, it is a good idea to train several networks to ensure that a network with good generalization is found. Furthermore, by retraining each network, one can verify a robust network performance.

Figure 5 shows the training procedure of 100 ANNs with architecture mentioned in Section III using the *trainbr* function for Bayesian regularization. Each ANN is trained starting from different initial weights and biases, and with different division for the training and test data sets. To ensure good generalization, each ANN is retrained 5 times.

```
net = fitnet(8, 'trainbr');
net.performFcn = 'mse';
numNN = 100;
numNNr = 5;
NN = cell(numNNr, numNN);
trace = cell(numNNr, numNN);
perfs = zeros(numNNr, numNN);
% train numNN ANNs
for i = 1:numNN
    % retrain each ANN numNNr times
    for j = 1:numNNr
      [NN{j}{i},trace{j}{i}] = train(net, x, t);
      y2 = NN{j}{i}(x2);
      perfs(j, i) = perform(NN{j}{i}, t2, y2);
      net = NN{j}{i};
    end
    % reinitialize initial weights and biases
    net = init(net);
end
minPerf = min(perfs(:))
[rowMin, colMin] = find(perfs == minPerf)
net = NN{rowMin}{colMin};
tr = trace{rowMin}{colMin};
```

Figure 5. Training 100 ANNs and retraining each ANN 5 times to find the best generalization.

The performance function, such as MSE, measures how well ANN can predict data, i.e., how well ANN can be generalized to new data. The test data sets are a good measure of generalization for ANNs since they are not used in training. A small performance function on the test data set indicates an ANN with good performance was found. In this work, the ANN with the lowest performance on the test data set is chosen to make future predictions.

Using the methodology described above, two ANNs are chosen to predict the gs energy and the gs point proton rms radius. The ANN prediction results for the gs energies and gs proton rms radii of ^{6}Li are presented in detail in this section. Comparison with the ab initio NCSM calculation results is also provided for the available data at $N_{\max} = 12 - 18$.

Figure 6 presents the gs energy of ^{6}Li as a function of the HO energy, $\hbar\Omega$, at selected values of the basis truncation parameter, $N_{\max}$. The dashed curves connect the NCSM calculation results using the Daejeon16 NN interaction for $N_{\max} = 2 - 10$, in increments of 2 units, used for ANN training and testing. The solid curves link the ANN prediction results for $N_{\max} = 12 - 70$. The sequence from $N_{\max} = 12 - 30$ is in increments of 2 units, while the sequence from $N_{\max} = 30 - 70$ is in increments of 10 units. The lowest horizontal line corresponds to $N_{\max} = 70$ and represents the nearly converged result predicted by ANN. Convergence is defined as independence of both basis space parameters, $\hbar\Omega$ and $N_{\max}$. The convergence pattern shows a reduction in the spacing between successive curves and flattening of the curves as $N_{\max}$ increases. The gs energy provided by the ANN decreases monotonically with increasing $N_{\max}$ at all values of $\hbar\Omega$. This demonstrates that the ANN is successfully simulating what is expected from theoretical physics. That is, in theoretical physics the energy variational principle requires that the gs energy behaves as a non-increasing function of increasing matrix dimensionality at fixed $\hbar\Omega$ and, furthermore, matrix dimension increases with increasing $N_{\max}$.

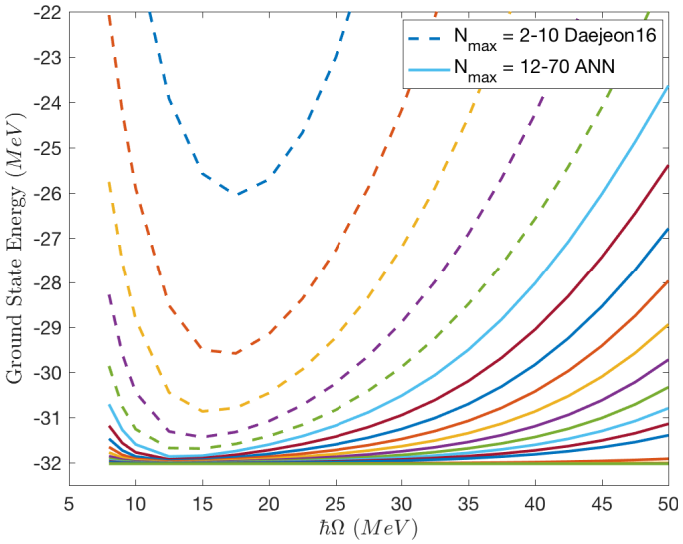

Figure 6. Calculated and predicted gs energy of ^{6}Li as a function of $\hbar\Omega$ at selected $N_{\max}$ values.

To illustrate the ANN prediction accuracy, the NCSM calculation results and the corresponding ANN prediction results of the gs energy of ^{6}Li are presented in Figure 7 as a function of $\hbar\Omega$ at $N_{\max} = 12, 14, 16$, and 18. The dashed curves connect the NCSM calculation results using the Daejeon16 NN interaction and the solid curves link the ANN prediction results. The nearly converged result predicted by ANN is also shown above the horizontal axis at $N_{\max} = 70$. Figure 7 shows good agreement between the calculated NCSM results and the ANN predictions up through $N_{\max} = 18$. Actual NCSM results always converged from above towards the exact result and become increasingly independent of the basis space parameters, $\hbar\Omega$ and $N_{\max}$. That the ANN result is essentially a flat line at $N_{\max} = 70$ and that the curves preceding it form an increasingly dense pattern approaching $N_{\max} = 70$ both provide indications that the ANN is producing a valid estimate of the converged gs energy.

The gs rms radii provide a very different quantity from NCSM results as they are found to be more slowly convergent than the gs energies and they are not monotonic. Figure 8 presents the calculated gs point proton rms radius of ^{6}Li as a function of $\hbar\Omega$ at selected values of $N_{\max}$. The dashed curves connect the NCSM calculation results using the Daejeon16 NN interaction up through $N_{\max} = 10$, while the solid curves link the ANN prediction results above $N_{\max} = 10$. The highest

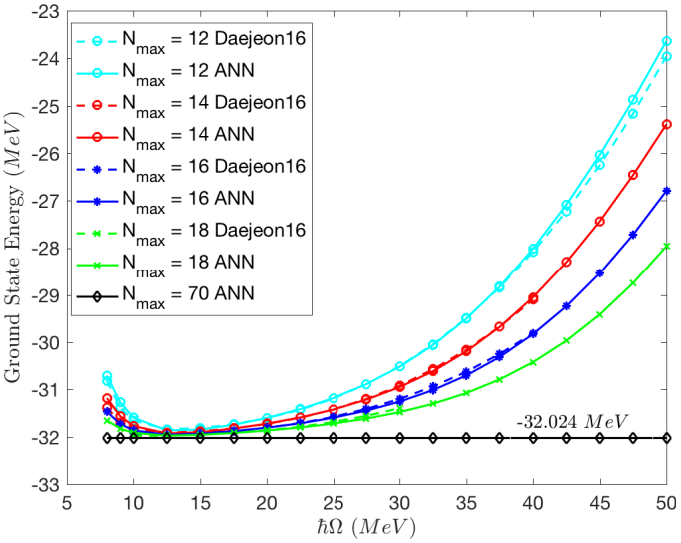

Figure 7. Comparison of the NCSM calculated and the corresponding ANN predicted gs energy values of ^{6}Li as a function of $\hbar\Omega$ at $N_{\max} = 12, 14, 16$, and 18. The lowest horizontal line corresponds to the ANN nearly converged result at $N_{\max} = 70$.

curve corresponds to $N_{\max} = 90$ and successively lower curves are obtained with $N_{\max}$ decreased by 10 units until the $N_{\max} = 30$ curve and then by 2 units for each lower $N_{\max}$ curve. The rms radius converges monotonically from below for most of the $\hbar\Omega$ range shown. More importantly, the rms radius shows the anticipated convergence to a flat line accompanied by an increasing density of lines with increasing $N_{\max}$. These are the signals of convergence that we anticipate based on experience in limited basis spaces and on general theoretical physics grounds.

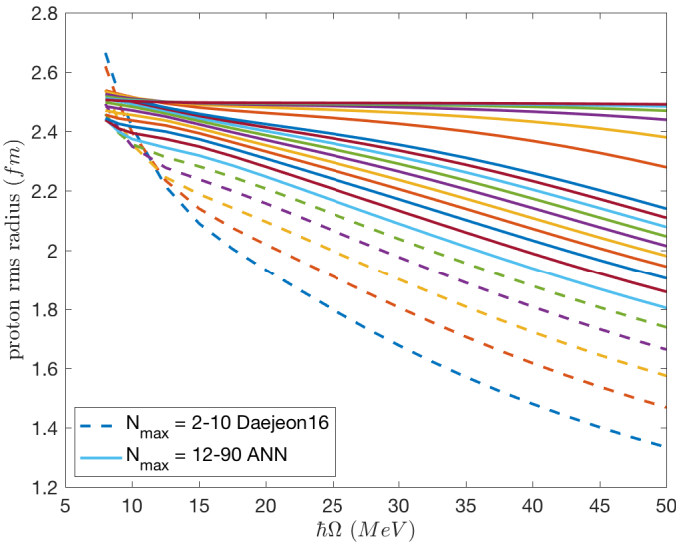

Figure 8. Calculated and predicted gs point proton rms radius of ^{6}Li as a function of $\hbar\Omega$ at selected $N_{\max}$ values.

The NCSM calculated values and the corresponding prediction values of the gs point proton rms radius of ^{6}Li are presented in Figure 9 for $N_{\max} = 12, 14, 16$, and 18. The dashed curves link the NCSM calculation results using the Daejeon16 NN interaction and the solid curves connect the ANN prediction results. As seen in this figure, the ANN predictions are in good agreement with the NCSM calculations, showing the efficacy of the ANN method.

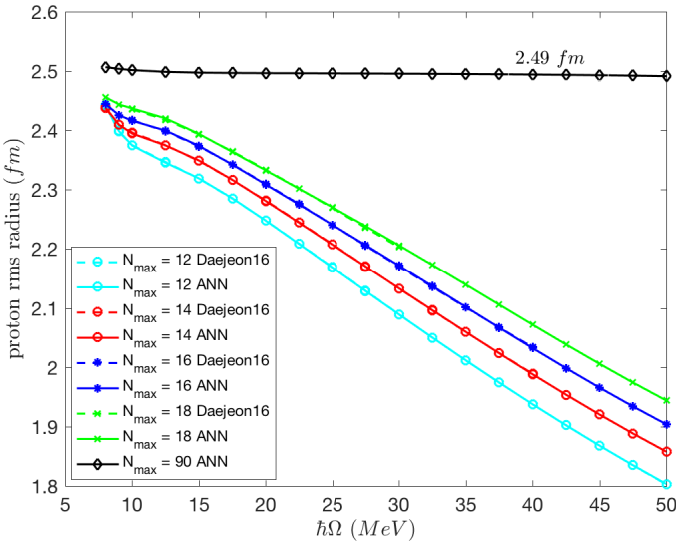

Figure 9. Comparison of the NCSM calculated and the corresponding ANN predicted gs point proton rms radius values of ^{6}Li as a function of $\hbar\Omega$ for $N_{\max} = 12, 14, 16$, and 18. The highest curve corresponds to the ANN nearly converged result at $N_{\max} = 90$.

Table I presents the nearly converged ANN predicted results for the gs energy and the gs point proton rms radius of ^{6}Li. As a comparison, the gs energy results from the current best theoretical upper bounds at $N_{\max} = 10$ and $N_{\max} = 18$ and from the Extrapolation B (Extrap B) method [34] at $N_{\max} \leq 10$ are provided. Similar to the ANN prediction, the Extrap B result arises when using all available results through $N_{\max} = 10$. The ANN prediction for the gs energy is below the best upper bound, found at $N_{\max} = 18$, which is about 85 KeV lower than the Extrap B result.

There is no extrapolation available for the rms radius, but we quote in Table I the estimated result by the *crossover-point* method [40] to be $\sim$ 2.40 fm. The *crossover-point* method takes the value at $\hbar\Omega$ in the table of rms radii results through $N_{\max} = 10$, which produces an rms radius result that is roughly independent of $N_{\max}$.

TABLE I. COMPARISON OF THE ANN PREDICTED RESULTS WITH RESULTS FROM THE CURRENT BEST UPPER BOUNDS AND FROM OTHER ESTIMATION METHODS.

Observable	Upper Bound $N_{\max} = 10$	Upper Bound $N_{\max} = 18$	Estimation[a] $N_{\max} \leq 10$	ANN $N_{\max} \leq 10$
gs energy (MeV)	-31.688	-31.977	-31.892	-32.024
gs rms radius (fm)	–	–	2.40	2.49

[a] The Extrap B method [34] for the gs energy and the crossover-point method [40] for the gs point proton rms radius

It is clearly seen from Figures 7 and 9 above that the ANN method results are consistent with the NCSM calculation results using the Daejeon16 NN interaction at $N_{\max} = 12, 14, 16$, and 18. Table I also shows that ANN's results are consistent with the best available upper bound in the case of the gs energy. The ANN's prediction for the converged rms radius is slightly larger than the result from the *crossover-point*

 ISBN: 978-1-61208-613-2

method and more consistent with the trends visible in Figure 9 at the higher $N_{\max}$ values. To measure the performance of ANNs, MSE for the training subsets up through $N_{\max} = 10$, as well as on the second test set for data at $N_{\max} = 12, 14, 16$, and 18, are provided in Table II.

TABLE II. THE MSE PERFORMANCE FUNCTION VALUES ON THE TRAINING AND TESTING DATA SETS AND ON THE $N_{\max} = 12, 14, 16$, AND 18 DATA SET.

Data Set	Whole Set $N_{\max} \leq 10$	Training Set $N_{\max} \leq 10$	Testing Set$_1$ $N_{\max} \leq 10$	Testing Set$_2$ $N_{\max} = 12 - 18$
gs energy (MeV)	4.86×10^{-4}	5.04×10^{-4}	3.80×10^{-4}	0.0072
gs rms radius (fm)	7.88×10^{-7}	4.49×10^{-7}	2.74×10^{-6}	9.24×10^{-7}

The small values of the performance function in Table II above indicate that ANNs with good generalizations were found to predict the results.

V. CONCLUSION AND FUTURE WORK

Feed-forward ANNs were used to predict the properties of the ^{6}Li nucleus such as the gs energy and the gs point proton rms radius. The advantage of the ANN method is that it does not need any mathematical relationship between input and output data. The architecture of ANNs consisted of three layers: two neurons in the input layer, eight neurons in the hidden layer and one neuron in the output layer. An ANN was designed for each output.

The data set from the ab initio NCSM calculations using the Daejeon16 NN interaction and basis spaces up through $N_{\max} = 10$ was divided into two subsets: 85% for the training set and 15% for the testing set. Bayesian regularization was used for training and doesn't require a validation set.

The designed ANNs were sufficient to produce results for these two very different observables in ^{6}Li from the ab initio NCSM. The gs energy and the gs point proton rms radius showed good convergence patterns and satisfy the theoretical physics condition, independence of basis space parameters in the limit of extremely large matrices. Comparisons of the results from ANNs with established methods of estimating the results in the infinite matrix limit are also provided. By these measures, ANNs are seen to be successful for predicting the results of ultra-large basis spaces, spaces too large for direct many-body calculations.

As future work, more Li isotopes such as ^{7}Li, ^{8}Li and ^{9}Li will be investigated using the ANN method and the results will be compared with results from improved extrapolation methods currently under development.

ACKNOWLEDGMENT

This work was supported by the Department of Energy under Grant Nos. DE-FG02-87ER40371 and DESC000018223 (SciDAC-4/NUCLEI). The work of A.M.S. was supported by the Russian Science Foundation under Project No. 16-12-10048. Computational resources were provided by the National Energy Research Scientific Computing Center (NERSC), which is supported by the Office of Science of the U.S. DOE under Contract No. DE-AC02-05CH11231. Personnel time for this project was also supported by Iowa State University.

REFERENCES

[1] P. Maris et al., "Origin of the Anomalous Long Lifetime of ^{14}C," Physical Review Letters, vol. 106, no. 20, May 2011, pp. 202 502–202 505, DOI: 10.1103/PhysRevLett.106.202502.

[2] B. R. Barrett, P. Navrátil, and J. P. Vary, "Ab Initio No Core Shell Model," Progress in Particle and Nuclear Physics, vol. 69, Mar 2013, pp. 131–181, DOI: 10.1016/j.ppnp.2012.10.003, ISSN: 0146-6410.

[3] S. C. Pieper and R. B. Wiringa, "Quantum Monte Carlo Calculations of Light Nuclei," Annual Review of Nuclear and Particle Science, vol. 51, no. 1, Dec 2001, pp. 53–90, DOI: 10.1146/annurev.nucl.51.101701.132506.

[4] K. Kowalski, D. J. Dean, M. Hjorth-Jensen, T. Papenbrock, and P. Piecuch, "Coupled Cluster Calculations of Ground and Excited States of Nuclei," Physical Review Letters, vol. 92, no. 13, Apr 2004, pp. 132 501–132 504, DOI: 10.1103/PhysRevLett.92.132501.

[5] W. Leidemann and G. Orlandini, "Modern Ab Initio Approaches and Applications in Few-Nucleon Physics with $A \geq 4$," Progress in Particle and Nuclear Physics, vol. 68, Jan 2013, pp. 158–214, DOI: 10.1016/j.ppnp.2012.09.001, ISSN: 0146-6410.

[6] D. Lee, "Lattice Simulations for Few- and Many-Body Systems," Progress in Particle and Nuclear Physics, vol. 63, no. 1, July 2009, pp. 117–154, DOI: 10.1016/j.ppnp.2008.12.001, ISSN: 0146-6410.

[7] E. Epelbaum, H. Krebs, D. Lee, and U. G. Meißner, "Ab Initio Calculation of the Hoyle State," Physical Review Letters, vol. 106, no. 19, May 2011, pp. 192 501–192 504, DOI: 10.1103/PhysRevLett.106.192501.

[8] A. M. Shirokov, A. I. Mazur, I. A. Mazur, and J. P. Vary, "Shell Model States in the Continuum," Physical Review C, vol. 94, no. 6, Dec 2016, pp. 064 320–064 323, DOI: 10.1103/PhysRevC.94.064320.

[9] A. Shirokov et al., "N3LO NN Interaction Adjusted to Light Nuclei in ab Exitu Approach," Physics Letters B, vol. 761, Oct 2016, pp. 87–91, DOI: 10.1016/j.physletb.2016.08.006, ISSN: 0370-2693.

[10] R. Machleidt and D. Entem, "Chiral Effective Field Theory and Nuclear Forces," Physics Reports, vol. 503, no. 1, June 2011, pp. 1–75, DOI: 10.1016/j.physrep.2011.02.001, ISSN: 0370-1573.

[11] A. Shirokov, J. Vary, A. Mazur, and T. Weber, "Realistic Nuclear Hamiltonian: Ab Exitu Approach," Physics Letters B, vol. 644, no. 1, Jan 2007, pp. 33–37, DOI: 10.1016/j.physletb.2006.10.066, ISSN: 0370-2693.

[12] P. Sternberg et al., "Accelerating Configuration Interaction Calculations for Nuclear Structure," in Proceedings of the 2008 ACM/IEEE Conference on Supercomputing – International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2008) Nov. 15–21, 2008, Austin, TX, USA. IEEE, Nov 2008, pp. 1–12, DOI: 10.1109/SC.2008.5220090, ISSN: 2167-4329, ISBN: 978-1-4244-2834-2.

[13] P. Maris, M. Sosonkina, J. P. Vary, E. Ng, and C. Yang, "Scaling of Ab-initio Nuclear Physics Calculations on Multicore Computer Architectures," Procedia Computer Science, vol. 1, no. 1, May 2010, pp. 97–106, ICCS 2010, DOI: 10.1016/j.procs.2010.04.012, ISSN: 1877-0509.

[14] H. M. Aktulga, C. Yang, E. G. Ng, P. Maris, and J. P. Vary, "Improving the Scalability of a Symmetric Iterative Eigensolver for Multi-core Platforms," Concurrency and Computation: Practice and Experience, vol. 26, no. 16, Nov 2014, pp. 2631–2651, DOI: 10.1002/cpe.3129, ISSN: 1532-0634.

[15] K. Hornik, M. Stinchcombe, and H. White, "Multilayer Feedforward Networks are Universal Approximators," Neural Networks, vol. 2, no. 5, Mar 1989, pp. 359–366, DOI: 10.1016/0893-6080(89)90020-8, ISSN: 0893-6080.

[16] C. M. Bishop, Neural Networks for Pattern Recognition. Oxford University Press, 1995, ISBN: 978-0198538646.

[17] S. Haykin, Neural Networks: A Comprehensive Foundation. Prentice-Hall Inc., 1999, Englewood Cliffs, NJ, USA, ISBN: 978-0132733502.

[18] S. Akkoyun, T. Bayram, S. O. Kara, and A. Sinan, "An Artificial Neural Network Application on Nuclear Charge Radii," Journal of Physics G: Nuclear and Particle Physics, vol. 40, no. 5, Mar 2013, pp. 055 106–055 112, DOI: 10.1088/0954-3899/40/5/055106.

[19] S. Athanassopoulos, E. Mavrommatis, K. A. Gernoth, and J. W. Clark, "One and two Proton Separation Energies from Nuclear Mass Systematics Using Neural Networks," Sep 2005, arXiv:0509075 [nucl-th].

[20] S. Athanassopoulos, E. Mavrommatis, K. Gernoth, and J. Clark, "Nuclear Mass Systematics Using Neural Networks," Nuclear

 ISBN: 978-1-61208-613-2

Physics A, vol. 743, no. 4, Nov 2004, pp. 222–235, DOI: 10.1016/j.nuclphysa.2004.08.006, ISSN: 0375-9474.

[21] C. David, M. Freslier, and J. Aichelin, "Impact Parameter Determination for Heavy-ion Collisions by use of a Neural Network," Physical Review C, vol. 51, no. 3, Mar 1995, pp. 1453–1459, DOI: 10.1103/PhysRevC.51.1453.

[22] S. A. Bass, A. Bischoff, J. A. Maruhn, H. Stöcker, and W. Greiner, "Neural Networks for Impact Parameter Determination," Physical Review C, vol. 53, no. 5, May 1996, pp. 2358–2363, DOI: 10.1103/PhysRevC.53.2358.

[23] F. Haddad et al., "Impact Parameter Determination in Experimental Analysis Using a Neural Network," Physical Review C, vol. 55, no. 3, Mar 1997, pp. 1371–1375, DOI: 10.1103/PhysRevC.55.1371.

[24] N. Costiris, E. Mavrommatis, K. A. Gernoth, and J. W. Clark, "A Global Model of β^-–Decay Half–Lives Using Neural Networks," Jan 2007, arXiv:0701096 [nucl-th].

[25] S. Akkoyun, T. Bayram, S. , and N. Yildiz, "Consistent Empirical Physical Formula for Potential Energy Curves of 38–66Ti Isotopes by Using Neural Networks," Physics of Particles and Nuclei Letters, vol. 10, no. 6, Nov 2013, pp. 528–534, DOI: 10.1134/S1547477113060022, ISSN: 1531-8567.

[26] "DIRAC Experiment," URL: http://www.cern.ch/DIRAC [accessed: 2018-01-17].

[27] "H1 Experiment," URL: http://www-h1.desy.de [accessed: 2018-01-17].

[28] R. Frühwirth, "Selection of Optimal Subsets of Tracks with a Feed-back Neural Network," Computer Physics Communications, vol. 78, no. 1–2, Dec 1993, pp. 23–28, DOI: 10.1016/0010-4655(93)90140-8, ISSN: 0010-4655.

[29] P. Abreu et al., "Classification of the Hadronic Decays of the Z^0 Into b and c Quark Pairs Using a Neural Network," Physics Letters B, vol. 295, no. 3–4, Dec 1992, pp. 383–395, DOI: 10.1016/0370-2693(92)91580-3, ISSN: 0370-2693.

[30] S. Abachi et al., "Direct Measurement of the top Quark Mass," Physical Review Letters, vol. 79, no. 7, Aug 1997, pp. 1197–1202, DOI: 10.1103/PhysRevLett.79.1197.

[31] B. Abbott et al., "Search for Scalar Leptoquark Pairs Decaying to Electrons and Jets in $\overline{p}p$ Collisions," Physical Review Letters, vol. 79, no. 22, Dec 1997, pp. 4321–4326, DOI: 10.1103/PhysRevLett.79.4321.

[32] D. H. Gloeckner and R. D. Lawson, "Spurious Center-of-Mass Motion," Physics Letters B, vol. 53, no. 4, Dec 1974, pp. 313–318, DOI: 10.1016/0370-2693(74)90390-6.

[33] B. N. Parlett, The Symmetric Eigenvalue Problem. Classics in Applied Mathematics, 1998, DOI: 10.1137/1.9781611971163, ISBN: 978-0-89871-402-9.

[34] P. Maris, J. P. Vary, and A. M. Shirokov, "Ab Initio No-Core Full Configuration Calculations of Light Nuclei," Physical Review C, vol. 79, no. 1, Jan 2009, pp. 014 308–014 322, DOI: 10.1103/PhysRevC.79.014308.

[35] M. T. Hagan and M. B. Menhaj, "Training Feedforward Networks with the Marquardt Algorithm," IEEE Transactions on Neural Networks, vol. 5, no. 6, Nov 1994, pp. 989–993, DOI: 10.1109/72.329697, ISSN: 1045-9227.

[36] D. J. MacKay, "Bayesian Interpolation," Neural Computation, vol. 4, no. 3, May 1992, pp. 415–447, DOI: 10.1162/neco.1992.4.3.415, ISSN: 0899-7667.

[37] D. W. Marquardt, "An Algorithm for Least-Squares Estimation of Nonlinear Parameters," Journal of the Society for Industrial and Applied Mathematics, vol. 11, no. 2, June 1963, pp. 431–441, SIAM, DOI: 10.1137/0111030, ISSN: 2168-3484.

[38] F. D. Foresee and M. T. Hagan, "Gauss-Newton Approximation to Bayesian Learning," in Proceedings of the International Joint Conference on Neural Networks, vol. 3. IEEE, Jun 1997, pp. 1930–1935, DOI: 10.1109/ICNN.1997.614194.

[39] G. Cybenko, "Approximation by Superpositions of a Sigmoidal Function," Mathematics of Control, Signals and Systems, vol. 2, no. 4, Dec 1989, pp. 303–314, DOI: 10.1007/BF02551274, ISSN: 1435-568X.

[40] S. K. Bogner et al., "Convergence in the No-Core Shell Model with Low-Momentum Two-Nucleon Interactions," Nuclear Physics A, vol. 801, no. 1, Mar 2008, pp. 21–42, DOI: 10.1016/j.nuclphysa.2007.12.008, ISSN: 0375-9474.

 ISBN: 978-1-61208-613-2

A Method for Recovering Speech Signals Heavily Masked by Music Based on the Affine Projection Algorithm

Robert Alexandru Dobre, Constantin Paleologu, Cristian Negrescu, and Dumitru Stanomir
Telecommunications Department
Politehnica University of Bucharest
Bucharest, Romania
email: rdobre@elcom.pub.ro, pale@comm.pub.ro, negrescu@elcom.pub.ro, dumitru.stanomir@elcom.pub.ro

***Abstract*—The importance of multimedia materials in justice is increasing. For example, a security camera recording could provide the evidence needed to clarify a given situation. The problems that arise are linked to the authenticity or intelligibility of the materials. There are situations in which the key material, (for example, a dialogue) is heavily masked. This paper presents the performances obtained by the Affine Projection Algorithm within a method for recovering speech signals masked by music. The results help in deciding if audio monitoring a certain acoustic environment could prove useful if the proposed method for extracting the speech is used afterwards.**

Keywords-multimedia forensic; noise reduction; adaptive filtering; affine projection algorithm.

I. INTRODUCTION

The rate at which multimedia materials are captured is increasing as the required technology nowadays can be fit into a smartphone. These recordings could prove to be important evidence in trials. But before they can be considered, they must be investigated to determine if they are the original versions and if the key element (image, video, sound) is clear. The domain that studies the methods that can be used to determine if a multimedia material is original or not is known as multimedia authentication and it is a subdomain of multimedia forensic. The other direction is represented by noise reduction, which has the main task to enhance the key element in an audio or video material. The contribution presented in this paper is part of the latter category and investigates the following situation: if suspects have to discuss something of great importance, it is very likely to do it in person. To decrease the chances to be intercepted (recorded), they could turn loud a nearby music system and the music would heavily mask their dialogue, making any recording gear placed in the room apparently useless. The masking melody can be identified thanks to software like Shazam. The signal recorded by the equipment placed in the room could be processed to subtract the musical part, revealing the dialogue. Even if the masking melody is identified and available, it cannot be subtracted directly because in the recording it appears affected by the acoustic environment (by the acoustic impulse response of the room). This is because the sound waves reflect on the walls of the room and other surfaces placed there (furniture, people, etc.) before arriving on the surface of the microphone and being recorded. The acoustic impulse response of the room can be modelled by a finite impulse response (FIR) filter. The method for extracting the speech signal is illustrated in Figure 1. The speech and the masking music signal propagate through the room and are captured by the microphone. If the original musical signal and the acoustic impulse response of the room are available, a replica of the recorded music signal can be obtained and subtracted from the recording, unveiling the dialogue. It can be considered the classical adaptive noise reduction configuration in which the musical signals play the role of two replicas of the same noise signal.

In Figure 1, $s_{\text{dialogue}}(t)$ represents the clean speech signal (without the effect of the room), and $n_{\text{melody}}(t)$ is the masking melody. The impulse response of the filter that models the acoustic environment of the room is $h(t)$ and $r(t)$ is the recorded signal, i.e., the sum of the clean signals affected by the acoustics of the room. The recorded signal is used to identify the masking melody. The heavier the masking, the easier the task of the music identification software. Having the identified song, one only needs the acoustic impulse response of the room to be able to reveal the dialogue. The adaptive algorithm used to estimate $h(t)$ is the affine projection algorithm (APA) because of its decent convergence speed and average computational complexity. An estimate for $s_{\text{dialogue}}(t)$ is the error signal of the algorithm, denoted by $e(t)$. The error signal will not be the clean speech signal, but the speech signal affected by the acoustics of the room. This effect is not problematic (if the acoustic environment is not heavily reverberant) because this is what it is heard naturally when one speaks in a room [1]. The paper investigates the effects of the length and sparsity of the impulse response of the filter (which models the acoustic environment) on the considered algorithm.

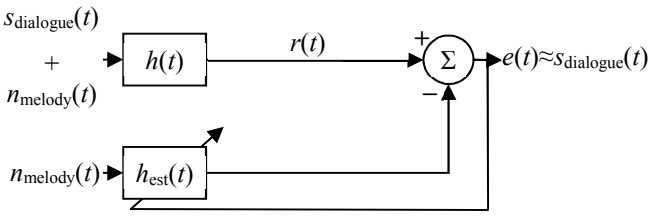

Figure 1. The adaptive noise reduction configuration modelling the real dialogue interception situation.

 ISBN: 978-1-61208-613-2

It is important to note that the system in Figure 1, which models the considered interception configuration, was described in continuous time, for simplicity. The adaptive filtering is a typical digital signal processing (DSP) application and all the results presented in this paper are obtained using DSP. The required operations to pass from continuous time modelling to the actual processing (sampling, quantization, etc.) do not need special attention as they do not introduce effects that should be considered, if properly done.

Besides this introduction, the paper consists of three sections as follows: Section II generally presents some key adaptive filtering notions, three adaptive algorithms, and the measures used to characterize the performance and impulse responses, Section III presents the experimental configuration and discusses the results, and Section IV concludes the paper.

II. ADAPTIVE FILTERING

An adaptive filter is a linear system whose impulse response is computed according to an optimization algorithm. The following descriptions are expressed in discrete time (where n is the time index) and only real signals are considered in this paper. An adaptive algorithm processes two signals, generally named in the literature as the input signal [denoted with $x(n)$] and the desired signal [denoted with $d(n)$], in a way that would minimize a cost function. Depending on the definition of the cost function, various adaptive algorithms exist. The method discussed in the paper uses the APA. A short description of the least-mean-squares (LMS) and the normalized LMS (NLMS) algorithms detailed in [2] and [3] will be presented further because it offers a better understanding of APA in particular, and of the adaptive filtering in general. Besides the aforementioned notations, in the equations will also be found the following: **w** – the adaptive filter's coefficients vector and e – the error signal, which are well-known notions in adaptive filtering literature.

A. The LMS and NLMS algorithms

The cost function in the case of the LMS algorithm gives the name of the algorithm. It is defined as:

$$C(n) = e^2(n) = \left[d(n) - y(n)\right]^2, \quad (1)$$

where $y(n)$ is the output of the adaptive filter. Minimizing the cost function with respect to the **w** vector gives the following update equation:

$$\mathbf{w}(n) = \mathbf{w}(n-1) + \mu\mathbf{x}(n)\left[d(n) - \mathbf{w}^{\mathrm{T}}(n-1)\mathbf{x}(n)\right], \quad (2)$$

where $\{\cdot\}^{\mathrm{T}}$ is the transposition operator and μ is the step size parameter. The values of μ that assure the convergence of the algorithm must respect the relation:

$$0 < \mu < \frac{2}{\mathrm{tr}\{\mathbf{R}\}}, \quad (3)$$

where **R** is the autocorrelation matrix of the input signal, which is given by:

$$\mathbf{R} = \mathrm{E}\left\{\mathbf{x}(n)\mathbf{x}^{\mathrm{T}}(n)\right\}, \quad (4)$$

$\mathrm{tr}\{\cdot\}$ represents the trace of a matrix, and $\mathrm{E}\{\cdot\}$ denotes mathematical expectation. The main advantage of the LMS algorithm is its simplicity, but equations (3) and (4) highlight its main problem, i.e., the values that assure the convergence are dependent on the input signal. This issue is solved in the NLMS algorithm in which the step size is scaled by the short time estimated power of the input signal. The update equation for the coefficients of the adaptive filter in the case of the NLMS algorithm becomes:

$$\mathbf{w}(n) = \mathbf{w}(n-1) + \frac{\mu\mathbf{x}(n)\left[d(n) - \mathbf{w}^{\mathrm{T}}(n-1)\mathbf{x}(n)\right]}{\mathbf{x}^{\mathrm{T}}(n)\mathbf{x}(n) + \delta}, \quad (5)$$

where δ is the regularization parameter, which avoids the division by zero (if the power of the input signal is estimated as zero), and

$$\mathbf{x}(n) = \left[x(n), x(n-1), \ldots, x(n-L+1)\right]^{\mathrm{T}}, \quad (6)$$

where L is the length of the adaptive filter. The step size that now assures the convergence of the algorithm can be chosen in the $0 < \mu < 2$ interval, independent on the data to be processed. Even if in the case of the NLMS algorithm the step size can be easily chosen, the disadvantage of this algorithm is its lack of flexibility (only one parameter – the step size – can be modified to get the desired behavior of the algorithm).

B. The affine projection algorithm

The APA [4] brings another degree of freedom in choosing the working parameters. Besides the step size [5] found also in the NLMS algorithm, a new "projection order" parameter (denoted by M) is introduced. It indicates how many input signal vectors [$\mathbf{x}(n)$] are used when computing the **w** vector. An $M \times L$ matrix is built using the M input signal vectors:

$$\mathbf{A}^{\mathrm{T}}(n) = \left[\mathbf{x}(n), \mathbf{x}(n-1), \ldots, \mathbf{x}(n-M+1)\right], \quad (7)$$

and equation (5) becomes:

$$\mathbf{w}(n) = \mathbf{w}(n-1) + \mu\mathbf{A}^{\mathrm{T}}(n)\left[\mathbf{A}(n)\mathbf{A}^{\mathrm{T}}(n) + \delta\mathbf{I}_M\right]^{-1}\mathbf{e}(n), \quad (8)$$

where $\mathbf{I}_M$ is the identity matrix of order M and, consequently:

$$\mathbf{e}(n) = \mathbf{d}(n) - \mathbf{y}(n), \quad (9)$$

 ISBN: 978-1-61208-613-2

$$\mathbf{d}(n) = \left[d(n), d(n-1), ..., d(n-M+1) \right]^{\mathrm{T}}, \quad (10)$$

$$\mathbf{y}(n) = \mathbf{A}(n)\mathbf{w}(n-1). \quad (11)$$

The downside of introducing this new parameter is an increase in computational complexity.

C. *Performance measurements of adaptive algorithms and sparsity degree of impulse responses*

In the problem stated in the introduction, the adaptive filter should estimate an unknown filter (the acoustic impulse response of the room). In the ideal event of a perfect estimation, the two filters would be identical. In real working conditions, perfect estimation is not likely to occur. In order to characterize how close the impulse response of the adaptive filter is to the impulse response to be estimated, a measure named "misalignment" (denoted with m) is introduced. Its computation is straightforward and, using the notations introduced in Figure 1, it can be written as:

$$m(n) = \left\| \mathbf{w}(n) - \mathbf{w}_{\text{t.b.e.}}(n) \right\|^2, \quad (12)$$

where $\mathbf{w}_{\text{t.b.e.}}(n)$ is the impulse response to be estimated and $\|\cdot\|$ is the l_2 norm.

Because of its large dynamic range, the misalignment is preferred to be expressed in dB. A misalignment as small as possible is desired. Another wanted behavior is that the misalignment should get to very small values in short time. The measure that qualitatively characterizes this property is the convergence speed (a high convergence speed is sought). The parameters of an adaptive algorithm should be tweaked to get the fastest convergence speed and the smallest steady-state misalignment. As seen in the previous subsection, greater flexibility comes at a cost of computational power.

A property of impulse responses which is of great importance especially in the case of acoustic systems is "sparsity". An impulse response is called "sparse" when only a small part of the values that compose it have notable values and others are insignificant. There are more ways in which the sparsity degree (denoted with χ) can be computed. In practice, good results are obtained using the following relation:

$$\chi(\mathbf{w}_{\text{t.b.e.}}) = \frac{L}{L-\sqrt{L}} \left(1 - \frac{\|\mathbf{w}_{\text{t.b.e.}}\|_1}{\sqrt{L}\,\|\mathbf{w}_{\text{t.b.e.}}\|} \right) \quad (13)$$

where $\|\cdot\|_1$ is the l_1 norm. The value returned by equation (13) can be between 1 and 0, the former indicating a high sparsity degree (there are only some dominant values in the analyzed vector). The effect of the sparsity degree on the behavior of the APA [6],[7] in the studied speech enhancement configuration is also investigated in the current paper.

III. EXPERIMENTAL RESULTS

Six impulse responses with various lengths and degrees of sparsity were used in the experiments. The considered impulse responses are illustrated in Figure 2 to Figure 7 and their degree of sparsity computed with equation (13) is mentioned. A speech and a musical signal were summed in a −40 dB signal-to-noise ratio (music in the role of noise) and then filtered with each of the presented impulse responses. Then the APA was used (with the original music signal as input and filtered mixture as desired signal) to estimate the acoustic

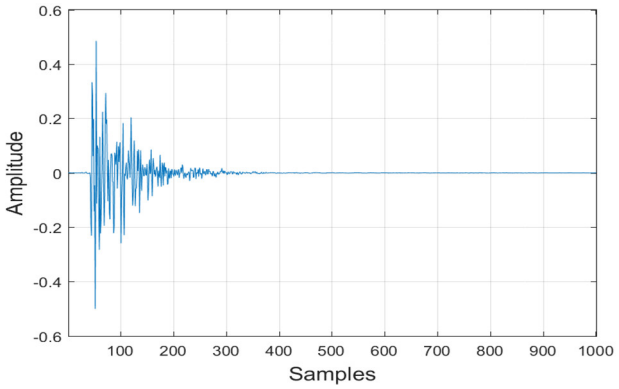

Figure 2. Acoustic impulse response with L=1001 and χ = 0.73852.

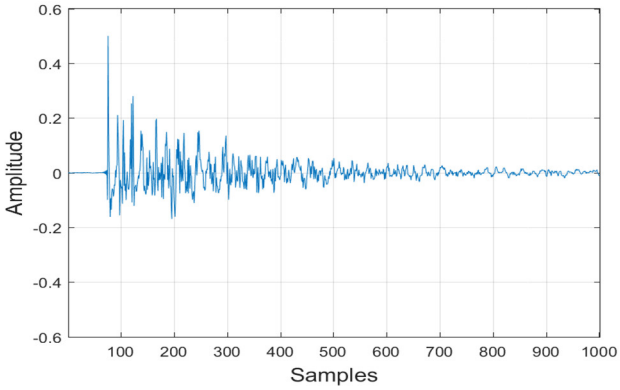

Figure 3. Acoustic impulse response with L=1001 and χ = 0.45617.

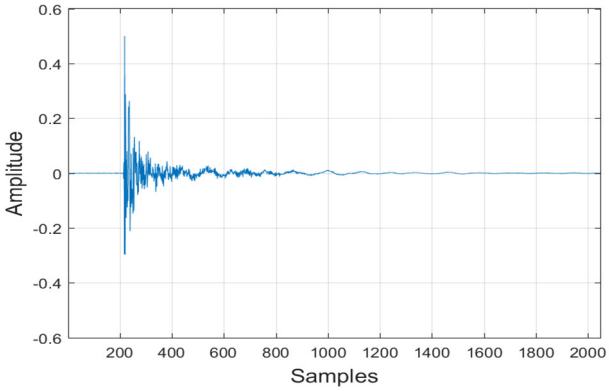

Figure 4. Acoustic impulse response with L=2048 and χ = 0.7344.

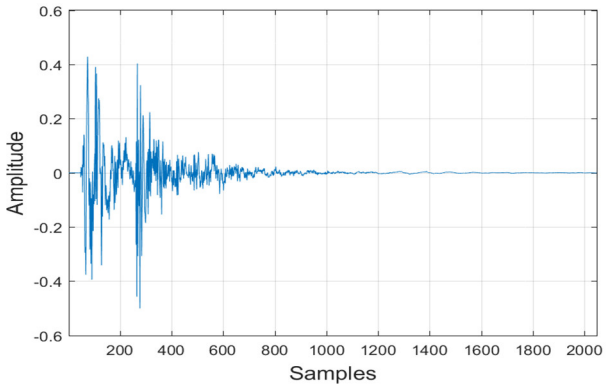

Figure 5. Acoustic impulse response with L=2048 and χ =0.64495.

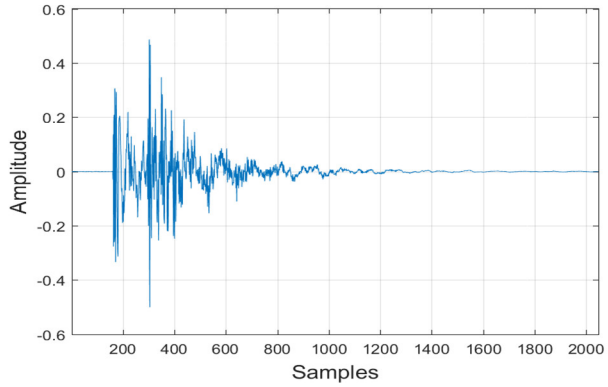

Figure 6. Acoustic impulse response with L=2048 and χ = 0.6085.

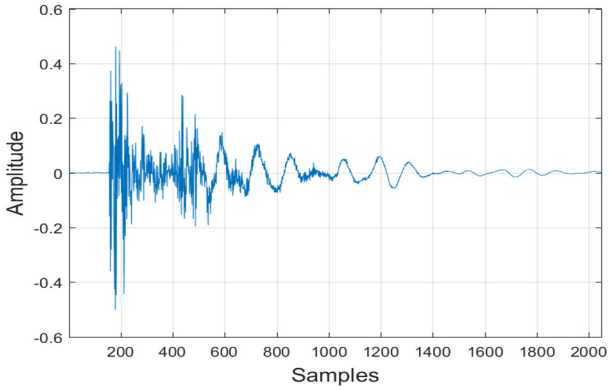

Figure 7. Acoustic impulse response with L=2048 and χ = 0.48781.

impulse response, measuring the performance with equation (12).

The situation presented in the introduction supposes that the acoustic environment [represented by $h(t)$] does not change in time. In real scenarios, this is very unlikely to happen because people would change their position, doors could be opened or closed etc. which would lead to a modification of the acoustic properties of the room. The duration of the signals used in the simulation was chosen to be 20 seconds. This provides a sufficient time to draw conclusions about the performances of the algorithm and keeps the simulation running time acceptable on most computers. A change in the impulse response that models the unknown filter was considered, implemented as an 8 samples time shift, after 10 seconds have passed. This is useful because it can highlight the ability of the algorithm to follow any changes that could occur in the acoustic properties of the room. The results are shown in Figure 8 to Figure 13 below.

The impulse responses that participated in the investigation have two lengths: 1001 (the ones illustrated in

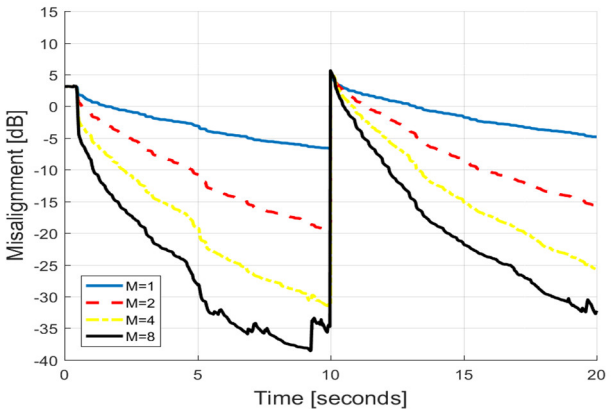

Figure 8. Misalignment of the APA when estimating the impulse response from Figure 2.

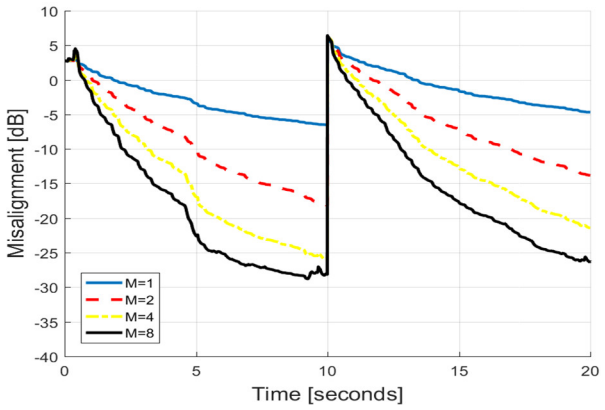

Figure 9. Misalignment of the APA when estimating the impulse response from Figure 3.

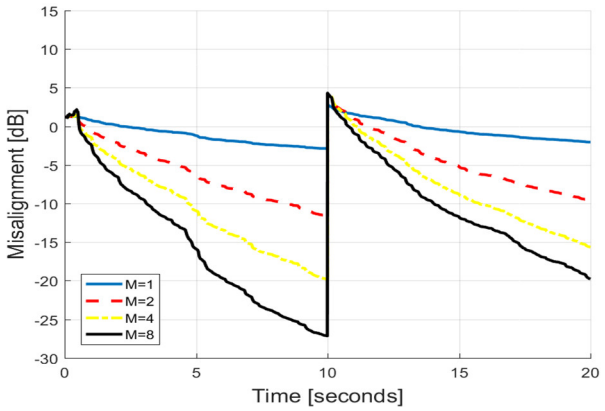

Figure 10. Misalignment of the APA when estimating the impulse response from Figure 4.

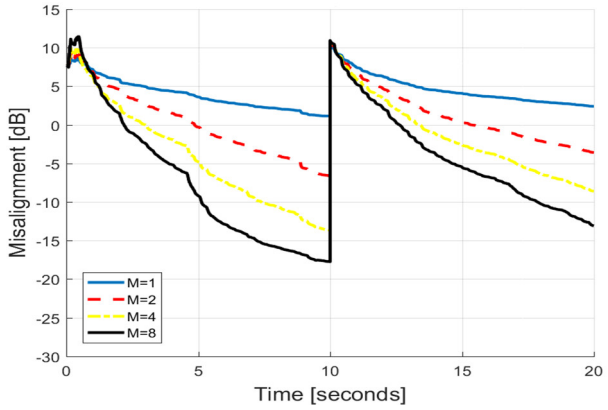

Figure 11. Misalignment of the APA when estimating the impulse response from Figure 5.

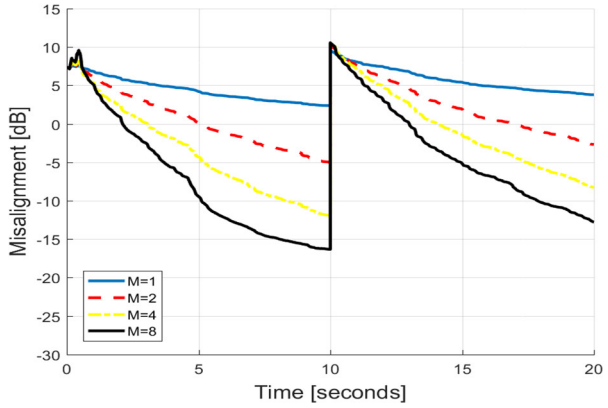

Figure 12. Misalignment of the APA when estimating the impulse response from Figure 6.

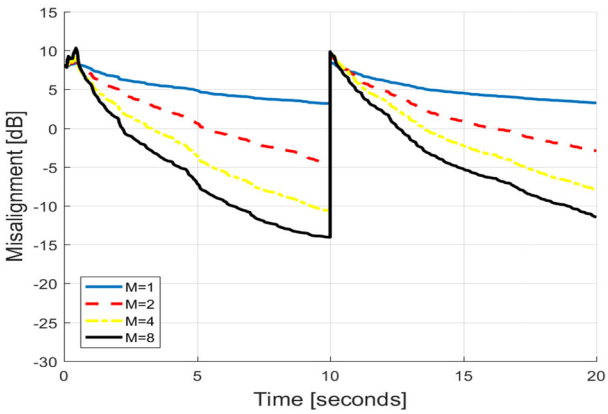

Figure 13. Misalignment of the APA when estimating the impulse response from Figure 7.

Figure 2 and Figure 3) and 2048 samples (shown in Figure 4 to Figure 7). This large difference helps identifying how the behavior of the proposed method based on the APA is affected by the length of the impulse responses. A longer impulse response has the significance of a more reverberant room (for example rooms with less furniture and very hard walls). Each of the two lengths category contains impulse responses with very different sparsity degrees. It can be seen, for example, that the impulse response illustrated in Figure 3 has the same length as the one in Figure 2, but a considerably smaller sparsity degree. This helps investigating in the same time the effect of two key properties of impulse responses (length and sparsity degree) on the considered method. The adaptive algorithm was run for projection orders equal to 1 (in this case the APA is equivalent with the NLMS algorithm and represents a typically used reference for the performance), 2, 4, and 8.

From the length point of view, the results are clear: the algorithm shows better results (lower misalignment) in the given simulation time for shorter impulse responses.

In the case of sparsity degree, the results show that sparse impulse responses lead to better performances. This can be observed for the impulse responses that have a length equal to 1001 samples, the first (shown in Figure 2) having a larger sparsity degree (0.73852) than the second one (Figure 3, sparsity degree equal to 0.45617), but also for the longer ones (the impulse responses in Figure 4 and Figure 7 have lengths equal to 2048 samples, but the sparsity degree of the first is equal to 0.7344 is greater than of the latter, 0.48781). Those results are shown in Figure 8 and Figure 9 for the first considered pair and in Figure 10 and Figure 13 for the second pair. The impulse responses illustrated in Figure 5 and Figure 6 have equal lengths and similar sparsity, so that the performances of the algorithm used by the forensic method were very similar in their cases (results shown in Figure 11 and Figure 12). The obtained graphs suggest that the method should be used if the room in which the intercepting device (microphone) is placed is small and not very reverberant.

It is of great importance to notice that the APA manages to obtain a misalignment less than −15 dB for all the impulse responses that were studied. It was determined that values for the misalignment greater than −10 dB lead to an unintelligible recovered speech signal. In the situations considered in this work, the best all-around results are obtained for a projection order equal to 8. In this case, the worst-case scenario is obtained when estimating the impulse response from Figure 7 (which has a large length and a relatively low sparsity degree). Up to 5 seconds of recovered speech signal could still be unintelligible (the time needed by the algorithm to get to −10 dB misalignment). For short and sparse impulse responses, the usage of a projection order larger than 4 does not bring an increase in performance to worth the extra cost of computational power.

It can be concluded that the APA based forensic method for recovering speech signals heavily masked by music is showing robustness properties and can be used when the recording was done in various acoustic environments. It also shows very good performances if the acoustic impulse response of the room is short and sparse (e.g., offices).

IV. Conclusion and Future Work

In this paper, the problem of recovering a speech signal heavily masked by music was described.

It was shown how a dialogue interception scenario can be modelled using adaptive filters (the adaptive noise reduction configuration). Short theoretical description of the LMS, NLMS, and APA helps to understand why the latter is a good

 ISBN: 978-1-61208-613-2

candidate to such signal processing method, thanks to its good performances, flexibility, and decent computational demands.

To evaluate the reliability of the method in various situations, a collection of six impulse responses with different lengths and sparsity degrees were used to simulate the acoustic environment in which the intercepting device was placed. To further increase the realism of the modelled scenario, a sudden change in the acoustic environment was introduced at the half of the investigation time, as an 8 samples time shift of the impulse response.

The results show that the method offers good performance especially for short and sparse impulse responses. In all the considered situations, the adaptive algorithm managed to obtain a misalignment equal or smaller than −15 dB, which indicates that the recovered signal has fair to high chances to be intelligible, confirming the versatility of the method. For short and sparse impulse responses, a projection order equal to 4 is recommended. In harsher situations, an M parameter equal to 8 could be needed to avoid getting a recovered speech signal with long unintelligible parts.

Since the effect of a change in the acoustic environment seems to be very clear (a large modification of the misalignment), new applications could be investigated in future works (e.g., monitoring of the acoustic environment).

ACKNOWLEDGMENT

This work was supported under the Grants SeaForest 86/2016, E-STAR 113/2016, and SenSyStar 190/2017.

REFERENCES

[1] R. A. Dobre, C. Negrescu, and D. Stanomir, “Development and testing of an audio forensic software for enhancing speech signals masked by loud music,” Advanced Topics in Optoelectronics, Microelectronics, and Nanotechnologies 2016, pp. 100103A-100103A-7, 2016.

[2] S. Haykin, Adaptive Filter Theory. Fourth Edition, Upper Saddle River, NJ:Prentice-Hall, 2002.

[3] A. H. Sayed, Adaptive Filters. New York, NY: Wiley, 2008.

[4] K. Ozeki and T. Umeda, "An adaptive filtering algorithm using an orthogonal projection to an affine subspace and its properties," Electron. Commun. Jpn., vol. 67-A, pp. 19-27, May 1984.

[5] C. Paleologu, J. Benesty, and S. Ciochina, "A variable step-size affine projection algorithm designed for acoustic echo cancellation," IEEE Trans. Audio, Speech, Language Processing, vol. 16, pp. 1466-1478, Nov. 2008.

[6] C. Paleologu, J. Benesty, and S. Ciochina, Sparse Adaptive Filters for Echo Cancellation. Morgan & Claypool Publishers, Synthesis Lectures on Speech and Audio Processing, 2010.

[7] C. Paleologu, S. Ciochina, and J. Benesty, "An efficient proportionate affine projection algorithm for echo cancellation," IEEE Signal Processing Lett., vol. 17, pp. 165-168, Feb. 2010.

 ISBN: 978-1-61208-613-2