Tomáš Horváth (Ed.)

Information Technologies - Applications and Theory

Conference on Theory and Practice of Information Technologies
Hotel Magura, Belianske Tatry, Slovakia, September 2012
Proceedings

ITAT 2012
Information Technologies – Applications and Theory

Conference on Theory and Practice of Information Technologies

Editor

Tomáš Horváth
Institute of Computer Science
Faculty of Science, P. J. Šafárik University
Šrobárova 2, 041 54 Košice, Slovak Republic

ISBN 978-80-971144-0-4

Published by Slovak Artificial Intelligence Society
Number of Pages: 68

Preface

The 12th conference **ITAT'12 Information Technologies – Application and Theory** was held at the hotel Magura located in Monková dolina, near Ždiar, Belianske Tatry, Slovakia in September 17-21th, 2012.

The conference is a traditional place of meetings for Czech and Slovak computer science communities. The emphasis is on exchange of ideas and information as well as on consolidation of bounds between the scientists of these two countries. Large space in the scientific and the social program is devoted to discussions. Conference languages are Slovak and Czech.

All the 25 submitted papers were reviewed by two independent reviewers. The proceedings consists of 8 selected scientific papers and the extended abstract of the invited talk.

The conference was co-organized by
– Institute of Computer Science, P. J. Šafárik University in Košice
– Faculty of Mathematics and Physics, Charles University in Prague
– Institute of Computer Science of Academy of Sciences of the Czech Republic, Prague
– Slovak Society for Artificial Intelligence

I would like to thank to the authors of presented papers, the invited speaker Jiří Kléma and all the reviewers for keeping the good scientific level of ITAT. Special thanks go to the organizing committee led by Peter Gurský for the great job in organization of the conference.

The conference was partially supported by the grant VEGA 1/0832/12. Special thanks go to our sponsor Profinit (`http://www.profinit.eu/`) profinit.
NEW FRONTIER GROUP

Tomáš Horváth

We recommend the use of Adobe Reader version 9.0 to view this pdf-file.

Program Committee

Tomáš Horváth, (Chair), *University of P.J. Šafárik, Košice, SK*
Radim Bača, *Technical University VŠB, Ostrava, CZ*
David Bednárek, *Charles University in Prague, Prague, CZ*
Mária Bieliková, *Slovak University of Technology, Bratislava, SK*
Jiří Dokulil, *Charles University in Prague, Prague, CZ*
Jana Dvořáková, *Charles University in Prague, Prague, CZ*
Peter Gurský, *University of P.J. Šafárik, Košice, SK*
Tomáš Holan, *Charles University in Prague, Prague, CZ*
Martin Holeňa, *Institute of Computer Science, AS CR, Prague, CZ*
Jozef Jirásek, *University of P.J. Šafárik, Košice, SK*
Jana Katreniaková, *Comenius University, Bratislava, SK*
Rastislav Královič, *Comenius University, Bratislava, SK*
Michal Krátký, *Technical University VŠB, Ostrava, CZ*
Věra Kůrková, *Institute of Computer Science, AS CR, Prague, CZ*
Markéta Lopatková, *Charles University in Prague, Prague, CZ*
Roman Neruda, *Institute of Computer Science, AS CR, Prague, CZ*
Dana Pardubská, *Comenius University, Bratislava, SK*
Tomáš Plachetka, *Comenius University, Bratislava, SK*
Martin Plátek, *Charles University in Prague, Prague, CZ*
Jaroslav Pokorný, *Charles University in Prague, Prague, CZ*
Karel Richta, *Charles University in Prague, Prague, CZ*
Gabriel Semanišin, *University of P.J. Šafárik, Košice, SK*
Vojtěch Svátek, *University of Economics, Prague, CZ*
Roman Špánek, *Institute of Computer Science, AS CR, Prague, CZ*
Július Štuller, *Institute of Computer Science, AS CR, Prague, CZ*
Peter Vojtáš, *Charles University in Prague, Prague, CZ*
Jakub Yaghob, *Charles University in Prague, Prague, CZ*
Filip Zavoral, *Charles University in Prague, Prague, CZ*

Organizing Committee

Peter Gurský, (chair), *University of P. J. Šafárik, Košice, SK*
Hanka Bílková, *Institute of Computer Science, AS CR, Prague, CZ*
Róbert Novotný, *University of P. J. Šafárik, Košice, SK*
Martin Šumák, *University of P. J. Šafárik, Košice, SK*
Lenka Pisková, *University of P. J. Šafárik, Košice, SK*

Organization

ITAT 2012 – Information Technologies – Applications and Theory was organized by
University of P. J. Šafárik, Košice, SK
Institute of Computer Science, AS CR, Prague, CZ
Faculty of Mathematics and Physics, Charles University in Prague, CZ
Slovak Society for Artificial Intelligence, SK

Table of Contents

Invited paper

Scientific papers

Machine learning applications in bioinformatics

Jiří Kléma

Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University
Technická 2, 166 27, Prague, Czech Republic
klema@labe.felk.cvut.cz,
WWW home page: http://labe.felk.cvut.cz/~klema/klema.html

Abstract. *Bioinformatics is a field of study dealing with methods for storing, retrieving and analyzing gene and protein oriented biological data. High-throughput technologies like DNA sequencing or microarrays allow researchers to obtain large volumes of heterogeneous and mutually interacting data. Analysis and understanding of these data provides a natural application field for machine learning algorithms. At the same time, bioinformatics is a scientific branch of such analytical complexity, data variety and abundance that it motivates further development of specialized learning algorithms such as co-clustering or multiple sequence alignment. This paper provides a brief overview of the topics and works discussed during my talk on machine learning applications in bioinformatics. The talk starts with a preview of fundamental bioinformatics analytical tasks solved by machine learning algorithms mentioning a few success stories. The second part summarizes the recent bioinformatics research carried out in my home research group, the Intelligent Data Analysis group of Czech Technical University.*

1 Analytical bioinformatics tasks

A complete overview of analytical bioinformatics tasks solvable and being solved by machine learning (ML) algorithms is out of scope of this short summary. [1] is a textbook that provides an introduction to the most important problems in computational biology and a unified treatment of the ML methods for solving these problems. The book is self-contained, its large part focuses on the principles of fundamental ML algorithms. A relevant concise review appeared in [2], its updated recent modification was presented in [3]. The reviews distinguish four principal classes of tasks. Firstly, a large group of bioinformatics problems can be posed as classification tasks. Genome annotation including gene finding and searching for DNA binding sites with proteins or gene function prediction and protein secondary structure prediction make examples. Secondly, clustering can be used to learn functional similarity from gene expression data or it can form phylogenetic trees. Thirdly, probabilistic graphical models can serve for modelling of DNA sequences in genomics or inference of genetic networks in systems biology. Last but not least, optimization algorithms have been proposed to solve the multiple sequence alignment problem or they appear in simplified models of protein folding.

1.1 Success stories and interactions

The bioinformatics tool with the largest impact is undoubtedly The Basic Alignment Search Tool (BLAST) and its successors [4] for searching a large sequence database against a query sequence. The NCBI server that provides the service with heuristic methods for sequence database searching handles more than half a million queries a day, the paper [4] introducing the improved PSI-BLAST has tens of thousands of citations. Another success story is an early case study on predictive classification from gene expression data [5]. The study proved feasibility of cancer classification based solely on gene expression monitoring. Although other latter studies showed that this positive result cannot be by means taken for granted, since then molecular classification is an option in disease diagnostics.

Bioinformatics directly motivates some cutting edge ML projects such as automated hypotheses generation and learning of optimal workflows. [6] reports the development of Robot Scientist "Adam", which autonomously generated functional genomics hypotheses about the yeast Saccharomyces cerevisiae and experimentally tested these hypotheses by using laboratory automation. One of its main objectives of the ongoing European ML and data mining project e-LICO [7, 8] is to implement an intelligent data mining assistant that takes in user specifications of the learning task and the available data, plans a methodologically correct learning process, and suggests workflows that the user can execute to achieve the prespecified objectives. Bioinformatics is the major application area.

2 IDA bioinformatics research topics

One of our main research topics is learning from gene expression data driven by background knowledge [9]. Mining patterns from gene expression data represents an alternative way to clustering [10]. Clustering provides the most straightforward and traditional approach to obtain co-expressed genes. However, a typical group of genes shares an activation pattern only under specific experimental conditions. Local methods such as pattern mining can identify exactly the sets of genes displaying a specific expression characteristic in a set of situations. The main bottleneck of this type of analysis is twofold – computational costs and an overwhelming number of candidate patterns which can hardly be further exploited. A timely application of background knowledge available in literature databases, gene ontologies and other sources can help to focus on the most plausible patterns only. Molecular classification of biological samples based on their gene-expression profiles is a natural learning task with immediate practical uses. Nevertheless, molecular classifiers based solely on gene expression in most cases cannot be considered useful decision-making tools or decision-supporting tools. Similarly to the domain of pattern mining, recent efforts in the field of molecular classification aim to employ background knowledge. The idea is to extract features that correspond to functionally related gene sets instead of the individual genes, respectively the probesets whose expression is available in the original expression data [11, 12].

The previous paragraph employs the available structural genomic knowledge to improve the analysis of gene expression data. We also studied several methods to create it from collections of free biomedical texts, namely the research papers and their short summaries [13]. [14] proposes a novel ball-histogram approach to DNA-binding propensity prediction of proteins.

Last but not least, the IDA group cooperates with several biological institutes and labs. To exemplify, [15] shows an application of the set-level approach discussed above to the particular domain of respirable ambient air particulate matter, the principal research partner was the Department of Genetic Ecotoxicology from Czech Academy of Sciences. [16] evaluates differences in the intragraft transcriptome after successful induction therapy using two rabbit antithymocyte globulins, the partner was the Department of Nephrology, Transplant Center, Institute for Clinical and Experimental Medicine.

References

1. P. Baldi,S. Brunak: *Bioinformatics: the machine learning approach*, 2nd edition, MIT Press, 2001, 452.
2. P. Larranaga, B. Calvo, R. Santana, C. Bielza, J. Galdiano, I. Inza, J. A. Lozano, R. Armananzas, G. Santafe, A. Perez, V. Robles: *Machine learning in bioinformatics.* Briefings in Bioinformatics, 7(1), 2005, 86–112.
3. I. Inza, B. Calvo, R. Armananzas, E. Bengoetxea, P. Larranaga, J. A. Lozano: *Machine learning: an indispensable tool in bioinformatics.* Methods Mol. Biol. 593, 2010, 25–48.
4. S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, D. J. Lipman: *Gapped BLAST and PSI-BLAST: A new generation of protein database search programs.* Nucleic Acids Research, 25, 1997, 3389–3402.
5. T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. , J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, E. S. Lander: *Molecular classification of cancer: class discovery and class prediction by gene expression monitoring.* Science, 286 (5439), 1999, 531–537.
6. R. D. King, J. Rowland, S. G. Oliver, M. Young, W. Aubrey, E. Byrne, M. Liakata, M. Markham, P. Pir, L. N. Soldatova, A. Sparkes, K. E. Whelan, A. Clare: *The automation of science.* Science 324 (5923), 2009, 85–89.
7. e-lico project: An e-laboratory for interdisciplinary collaborative research in data mining and data-intensive science, `http://www.e-lico.eu/`, August 2012.
8. M. Hilario, P. Nguyen, H. Do, A. Woznica, A. Kalousis: *Ontology-based meta-mining of knowledge discovery workflows.* In Jankowski, N., Duchs, W. Grabczewski, K., Meta-Learning in Computational Intelligence, Springer, 2011, 273–316.
9. J. Klema: *Learning from heterogeneous genomic data.* FEE CTU, habilitation thesis, to appear.
10. J. Klema, S. Blachon, A. Soulet, B Cremilleux, O. Gandrilon: *Constraint-based knowledge discovery from SAGE data.* In Silico Biology, 8, 0014, 2008.
11. M. Holec, J. Klema, F. Zelezny, J. Tolar: *Comparative evaluation of set-level techniques in predictive classification of gene expression samples.* BMC Bioinformatics, 13, (10), 2012, S15.
12. M. Krejnik, J. Klema: *Empirical evidence of the applicability of functional clustering through gene expression classification.* IEEE/ACM Transactions on Computational Biology and Bioinformatics, 9(3), 2012, 788–798.
13. M. Plantevit, T. Charnois, J. Klema, C. Rigotti, B. Cremilleux: *Combining sequence and itemset mining to discover named entities in biomedical texts: A new type of pattern.* International Journal of Data Mining, Modelling and Management, 1(2), 2009, 119–148.
14. A. Szaboova, O. Kuzelka, F. Zelezny, J. Tolar: *Prediction of DNA-binding propensity of proteins by the ball-histogram method using automatic template search BMC.* Bioinformatics 13 (10), 2012, 3.
15. H. Libalova, K. Uhlirova, J. Klema, M. Machala, R. Sram, M. Ciganek, J. Topinka: *Global gene expression changes in human embryonic lung fibroblasts induced by organic extracts from respirable air particles.* Particle and Fibre Toxicology, 9(1), 2012.
16. M. Urbanova, I. Brabcova, E. Girmanova, F. Zelezny, O. Viklicky: *Differential regulation of the nuclear factor-kappa-B pathway by rabbit antithymocyte globulins in kidney transplantation.* Transplantation 93(6), 2012, 589–96.

RBF-based surrogate model for evolutionary optimization*

Lukáš Bajer[1,2] and Martin Holeňa[2]

[1] Faculty of Mathematics and Physics, Charles University
Malostranské nám. 25, Prague 1, Czech Republic
bajer@cs.cas.cz

[2] Institute of Computer Science, Academy of Sciences of the Czech Republic
Pod Vodárenskou věží 2, Prague 8, Czech Republic
martin@cs.cas.cz

Abstract. *Many today's engineering tasks use approximation of their expensive objective function. Surrogate models, which are frequently used for this purpose, can save significant costs by substituting some of the experimental evaluations or simulations needed to achieve an optimal or near-optimal solution. This paper presents a surrogate model based on RBF networks. In contrast to the most of the surrogate models in the current literature, it can be directly used for problems with mixed continuous and discrete variables – clustering and generalized linear models are employed for dealing with discrete covariates. The model has been tested on a benchmark optimization problem and its approximation properties are presented on a real-world application data.*

1 Introduction

Optimization of different kinds of empirical objective functions is included in many of todays engineering or industrial applications – in situations where the value of the objective function is obtained through some measurement, experiment or simulation. High costs or extensive time demands needed for evaluating such functions motivate engineers to reduce the number of such evaluations.

Surrogate modelling [3, 6] is a popular approach which substitutes an approximating model for some of the original function evaluations. This concept is widely used in connection with evolutionary algorithms (EAs). Here, some of the individuals are assessed with not necessary accurate, but much faster model. This brings an important benefit: a notably larger population can be evolved in parallel. Even though the precise evaluation can be made only on a limited number of individuals, the EA can explore a larger part of the input space.

Lots of current literature covers optimization in continuous, or in discrete domains. However, the area of industrial optimization is often characterized by both continuous and discrete variables [16, 7]. This paper describes a particular surrogate model based on radial basis function (RBF) networks and generalized linear models (GLMs). Most of the existing works [22, 17, 9] deal with only continuous domains or combination with integer variables, but the works dealing with mixed-variables surrogate models are rather few [21, 19].

In our model, multiple RBF networks are trained and discrete variables are used either for focusing training of the networks on the most appropriate data, or generalized linear model is constructed on this part of the data.

The paper is organized as follows: in the next section, we recall principles of surrogate modelling, RBF networks and GLMs. Section 3 describes our approach to constructing a surrogate models and using it in optimization. Finally, Section 4 provides the results of testing on a benchmark function and real-world data.

2 Problem description

For any given objective function $f : \mathbf{S} \to \mathbb{R}$, we consider the *mixed-variable optimization problem* (maximization) as finding the global optimum $\mathbf{x}^\star = (x_1^{(\mathrm{C})}, \ldots, x_n^{(\mathrm{C})}, x_1^{(\mathrm{D})}, \ldots, x_d^{(\mathrm{D})}) \in \mathbf{S}$ such that

$$f(\mathbf{x}^\star) = \max_{\mathbf{x} \in \mathbf{S}} f(\mathbf{x}). \tag{1}$$

The search space $\mathbf{S}$ has of n continuous and d discrete variables; forming corresponding subspaces $\mathbf{S}^{(\mathrm{C})}$ and $\mathbf{S}^{(\mathrm{D})}$. In addition, we suppose that the value sets $V_s(X_i^{(\mathrm{D})})$, $i = 1, \ldots, d$ of the discrete variables are finite and we do not distinguish between ordinal or nominal categorical variables – we assume no ordering on any of the $V_s(X_i^{(\mathrm{D})})$.

2.1 Involved methods

Surrogate modelling. Approximation of the fitness function with some regression model is a common cure for tasks when empirical objective function has to be used. These *surrogate models* simulate behaviour of

* This work was supported by the Grant Agency of the Charles University (GA UK), grant number 278511/2011 (Lukáš Bajer), and by the Czech Science Foundation (GA CR), grant number 201/08/0802 (Martin Holeňa).

the original function while being much cheaper and much less time consuming to evaluate.

As a surrogate model, mainly nonlinear regression models are used, for example gaussian processes [4] or artificial neural networks. In connection with evolutionary optimization, neural networks of the type multilayer perceptrons [10] and networks with radial basis functions [22, 17] have been particularly popular. The last mentioned kind of neural networks underlies also the model reported in this paper.

Combining of the original fitness function and the surrogate model is determined by *evolution control* (EC). In the literature [10], individual and generation based approaches are distinguished. While the individual-based EC chooses for evaluation by the original fitness only part of an enlarged population, the generation-based approach evaluates in different generations the whole population by either the original, or the model fitness.

RBF networks compute a mapping from the input space (typically a subspace of $\mathbb{R}^n$) to $\mathbb{R}$ (for simplicity we will focus on versions with scalar output) [5]. The mapping can be expressed as

$$f(\mathbf{x}) = \sum_{i=1}^{g} \pi_i f_i(||\mathbf{x} - \mathbf{c}_i||) \tag{2}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the input, $g \in \mathbb{N}$ the number of components, $f_i : \mathbb{R}^n \to \mathbb{R}$ are radial basis functions, $\pi_i \in \mathbb{R}$ their weights, $\mathbf{c}_i \in \mathbb{R}^n$ radial functions' centres, and $||.||$ is a norm. As functions f_i, Gaussian functions with scalar width δ_i and euclidean norm $f_i(\mathbf{x}; \mathbf{c}_i, \delta_i) = e^{-\delta_i ||\mathbf{x}-\mathbf{c}_i||^2}$ are used most commonly.

Generalized linear models are a natural generalization of classical linear regression models [13]. They consist of three parts: (1) the *random component* – independent observed values $\mathbf{Y}$ following a distribution from the exponential family with mean $E(\mathbf{Y}) = \boldsymbol{\mu}$ and constant variance σ^2, (2) the *systematic component* which relates values of explanatory (input) variables $(\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^d)$ through a linear model with parameters $\beta_1, \ldots, \beta_d$

$$\boldsymbol{\eta} = \sum_{j=0}^{d} \mathbf{x}^j \beta_j \tag{3}$$

to a *linear predictor* $\boldsymbol{\eta}$, and (3) a *link function* g that connects the random and systematic components together: $\boldsymbol{\eta} = g(\boldsymbol{\mu})$. The explanatory variables are usually supplemented with the constant vector of ones corresponding to an intercept parameter β_0.

GLMs are particularly useful for our work because they are able to express a relation between discrete (integer or after a transformation of values even nominal) input variables and a continuous response.

3 Our strategy for using surrogate-assisted genetic optimization

Our version of the surrogate-assisted genetic algorithm including a detailed pseudo-code has been introduced in the previous article [1]. This section describes the construction and using of surrogate models based on RBF networks.

3.1 Model construction

RBF networks, which were defined in Section 2.1, enable us to use only continuous variables for their fitting. Construction of our first surrogate model [1] starts with clustering of the available training data according to their discrete values into several clusters in order to focus the RBF networks training on the most similar datapoints. Let us call this model *RBF/discrete clustering*, or shortly *RBF/DSCL model*. Subsequently, separate RBF networks are fitted with the data of each such a cluster using the datapoints' continuous variables. The algorithm is the same as described on the Fig. 1 except the omitted steps (1)–(3), and the clustering which is made using discrete values from the training database $\mathbf{D}$ in the step (4).

This approach does not utilize relation between values of the discrete input variables and the response variable. As was stated in Section 2.1, such a relation can be expressed by generalized linear models, and these models form an important part of our new *RBF/GLM* surrogate *model*.

Training the RBF/GLM model starts with construction of two auxiliary models: the first, *global* RBF network $\hat{f}_{\text{RBF}} : \mathbf{S}^{(\text{C})} \to \mathbb{R}$ is fitted on the continuous input variables while the second, GLM $\hat{f}_{\text{GLM}} : \mathbf{S}^{(\text{D})} \to \mathbb{R}$ is built using the discrete variables. Both of them make use of all the available training data and regress the response-variable values.

Global RBF network. Training of the auxiliary RBF network works similarly to the training of the RBF networks in the previous RBF/DSCL model [1] – the same starting values for centers and weights, and cross-validation for choosing the best number of components g is used. However, instead of clusters, all the data in the database $\mathbf{D}$ are used at once.

GLM model. Generalized linear model is used in its continuous-response form and responses are supposed from normal distribution $\mathbf{Y} \sim N(\boldsymbol{\mu}, \sigma^2)$. Even though the latter assumption generally does not hold, GLMs still provide useful mean of regression expressed on the basis of the discrete values.

Before using or fitting the GLM, the discrete values must be converted to a proper representation. Since we do not expect any ordering on the discrete values, we have chosen *dummy coding* [13] which establishes one binary indicating variable $I_{ij} \in \{0,1\}$ for each nominal value from the value sets $V_s(X_i^{(\mathrm{D})})$, $i = 1, \ldots, d$, $j = 1, \ldots, |V_s(X_i^{(\mathrm{D})})|$ of the original discrete variables. Assignment between the original discrete values and the dummy coding

$$\mathit{dummy} : \mathbf{S}^{(\mathbf{D})} \to \{0,1\}^{|V_s(X_1)|+\ldots+|V_s(X_d)|} \qquad (4)$$

has to be recorded for evaluation with the surrogate model.

Final RBF clustered model. Having created the *global* RBF network $\hat{f}_{\mathrm{RBF}}$ and the GLM model $\hat{f}_{\mathrm{GLM}}$, we can proceed with the construction of the *final* RBF clustered surrogate model $\hat{f} : \mathbf{S}^{(\mathrm{C})} \to \mathbb{R}$. The process starts with clustering of the training data from the database $\mathbf{D} = \{\mathbf{x}_i^{(\mathrm{D})}, \mathbf{x}_i^{(\mathrm{C})}, y_i\}_{i=1}^{N}$ according to the difference between responses of the two auxiliary models on the corresponding input variables (for $i = 1, \ldots, N$)

$$\mathit{diff}_i = \hat{f}_{\mathrm{RBF}}(\mathbf{x}_i^{(\mathrm{C})}) - \hat{f}_{\mathrm{GLM}}(\mathit{dummy}(\mathbf{x}_i^{(\mathrm{D})})). \qquad (5)$$

The sizes of the clusters have to be at least $s_{\min}$ – the minimal number of data needed for fitting one RBF network. This number is provided by the user and its best value depends on a particular task. The higher the $s_{\min}$ is, the more components can each RBF network have, but the more distinct discrete values are usually grouped together in one cluster.

```
FitTheModel($s_{\min}$, $\mathbf{D}$, $e$)
Arguments: $s_{\min}$ – min. size of clusters,
    $\mathbf{D}$ – database, $e$ – type of error estimate:
    MSE, AIC, or BIC
Steps of the procedure:
  (1) $(\hat{f}_{\mathrm{RBF}}, \mathit{rbf}_{\mathrm{GLOB}}) \leftarrow$ fit the global RBF
  (2) $(\hat{f}_{\mathrm{GLM}}, \mathit{glm}) \leftarrow$ fit the GLM
  (3) $\{\mathit{diff}_i\}_{i=1}^{N} \leftarrow$ differences $(\hat{f}_{\mathrm{RBF}} - \hat{f}_{\mathrm{GLM}})$ on $\mathbf{D}$
  (4) $\{C_j\}_{j=1}^{m} \leftarrow$ cluster $\mathbf{D}$ into clusters of size
        at least $s_{\min}$ according to $\{\mathit{diff}_i\}_{i=1}^{N}$
  (5) for each cluster $C_j$, $j = 1, \ldots, m$
  (6)     for $g_j = 1, \ldots, g_j^{\max}$
  (7)         $\mathit{mse}[j, g_j] \leftarrow$ average $\mathrm{MSE}_{CV}$ from
                fitting RBF with $g_j$ components
  (8)     $g_j^{\star} \leftarrow$ the number of components
            of the best RBF
  (9)     $\mathit{rbf}_j \leftarrow$ retrained RBF network
            with $g_j^{\star}$ components
 (10)     $\mathit{mse}_j \leftarrow \mathit{mse}[j, g_j^{\star}]$
Output: $\{\mathit{rbf}_{\mathrm{GLOB}}, \mathit{glm}, (\mathit{rbf}_j, \mathit{mse}_j, \mathit{diff}_j)_{j=1}^{m}\}$
```

Fig. 1. Pseudo-code of the fitting procedure.

One separate RBF network rbf_j is trained on the data of each cluster C_j, $j = 1, \ldots, m$. The maximal number of components of each network is upper-bounded by $g_j^{\max} = \lfloor (\frac{k-1}{k}|C_j|)/\rho \rfloor$. Training these networks is analogous to training of the *global* RBF network described in Section 3.1. The only difference is in the training data: only the data of individual clusters are used for each network.

3.2 Evaluation with the surrogate model

Once the surrogate model is built, it can be used for evaluating individuals resulting from the evolution. The parameters of the model can be summarized as $\{\mathit{rbf}_{\mathrm{GLOB}}, \mathit{glm}, (\mathit{rbf}_j, \mathit{mse}_j, \mathit{diff}_j)_{j=1}^{m}\}$. Here, $\mathit{rbf}_{\mathrm{GLOB}}$ are *global* RBF network parameters, $\mathit{glm} = (\beta_0, \ldots, \beta_r)$ are parameters of the GLM, rbf_j global RBF network parameters, mse_j are the MSE_{CV} obtained from cross-validation, and diff_j are the difference *diff* (5) averaged on the j-th cluster's data.

Given a new individual $(\tilde{\mathbf{x}}^{(\mathrm{C})}, \tilde{\mathbf{x}}^{(\mathrm{D})})$, evaluation with the surrogate model starts with computing the difference between responses of the *global* RBF network and GLM with corresponding parameters $\mathit{rbf}_{\mathrm{GLOB}}$ and *glm*

$$\begin{aligned}\widetilde{\mathit{diff}} = {} & \hat{f}_{\mathrm{RBF}}(\tilde{\mathbf{x}}^{(\mathrm{C})}; \mathit{rbf}_{\mathrm{GLOB}}) \\ & - \hat{f}_{\mathrm{GLM}}(\mathit{dummy}(\tilde{\mathbf{x}}^{(\mathrm{D})}); \mathit{glm}). \end{aligned} \qquad (6)$$

Based on this value, the index c of the cluster with the average difference most similar to the individual's difference is obtained

$$c = \arg \min_{j=1,\ldots,m} |\mathit{diff}_j - \widetilde{\mathit{diff}}|. \qquad (7)$$

Finally, the response of the c-th *final* RBF network is used as a return value of the surrogate model

$$\tilde{y} = \hat{f}(\tilde{\mathbf{x}}^{(\mathrm{C})}; \mathit{rbf}_c) = \sum_{i=1}^{g_c^{\star}} \pi_{ic} f_{ic}(||\tilde{\mathbf{x}}^{(\mathrm{C})} - \mathbf{c}_{ic}||). \qquad (8)$$

If more than one cluster is at the same distance from the individual, the RBF network with the lowest MSE_{CV} is chosen.

4 Implementation and results of testing

Our algorithms were implemented in the MATLAB environment. We have been utilizing the Global Optimization Toolbox which provided us with a platform for testing the model on a benchmark optimization task. Similarly, our hierarchical clustering method extends the cluster analysis from the Statistical Toolbox which provide us with GLM fitting procedure, too, and we employ a nonlinear curve-fitting from the Optimization Toolbox for fitting RBF networks.

4.1 Model fitting

Our models have been tested on three different kinds of data. The first two datasets (Valero and HCN) are the same as in our last article [1], the third is the building1 dataset from Proben1 [18] collection.

Valero's [20] benchmark fitness function was constructed to resemble empirical fitness functions from chemical engineering. The surrogate models have been 10-times trained on dataset with 2000 randomly generated data. Using the same settings for fitting, the average root of the MSE (RMSE) of the new RBF/GLM model has been only slightly decreased. (see Table 1 and the top graphs on Fig. 2).

Valero	RBF/GLM	**14.046** ± 1.0435
	RBF/DSCL	14.499 ± 1.518
HCN	RBF/GLM	**10.340** ± 1.866
	RBF/DSCL	15.620 ± 1.519
building1	RBF/GLM	**0.06407** ± 0.00496
	RBF/DSCL	0.13618 ± 0.00455

Table 1. Surrogate-models' average regression RMSE on Valero's benchmark, HCN data and building1 dataset.

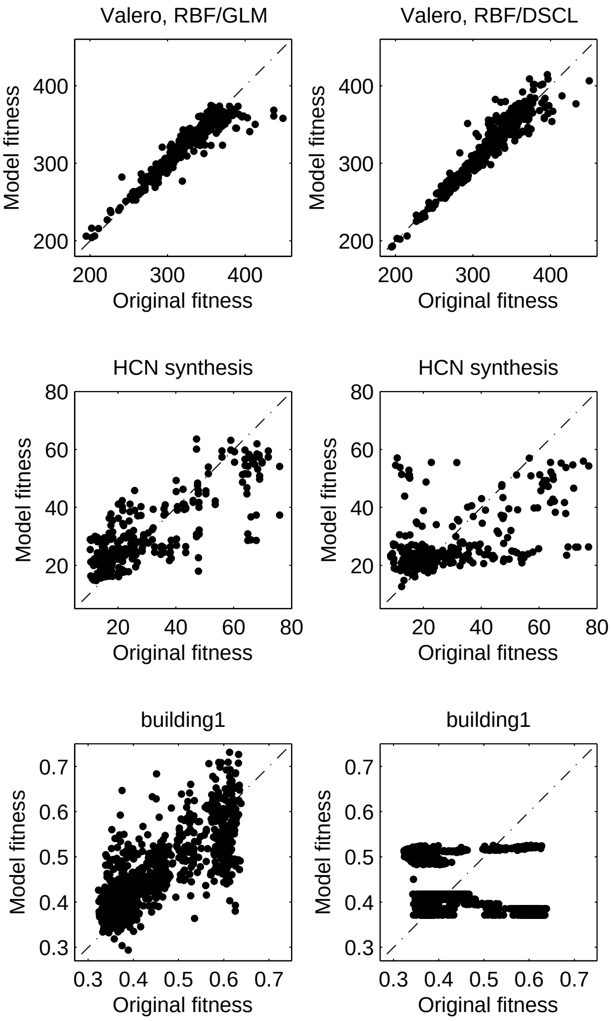


Fig. 2. Scatter plots of the RBF/GLM (left column) and RBF/DSCL model (right column) on testing data.

The second dataset is from a real application in chemical engineering (cf. using RBF networks in this application area e.g. in the work of Jun [11]): the optimization of chemical catalysts for Hydrocyanic acid (HCN) synthesis [14]. Solutions of this task are composed of two discrete and 11 continuous variables, the whole dataset has 696 items. Fitting results are substantially different from the benchmark problem (considering the average response in the dataset $\bar{y} = 31.17$, the measured RMSE's are relatively much higher: see middle row of graphs on Fig. 2). RMSE of the new RBF/GLM model has been decreased by nearly 35 % comparing to the previous model's error.

Prechelt's Proben1 [18] is a popular collection of datasets for data mining, originally intended for neural networks benchmarking. We have tested our models on the building1 dataset using the first response variable indicating electrical energy consumption in a building; multiple-trained on the first 3156 and tested on the remaining 1052 data, as suggested by Prechelt. Average results from ten trainings show that the former RBF/DSCL model is not able to sufficiently express the relation between discrete variables and the output. Conversely, the results of the RBF/GLM model are at least comparable to the results reported elsewhere [12, 2, 15].

4.2 Genetic algorithm performance on the benchmark fitness

The benchmark fitness enabled us to test the model with the GA [1]. As shown in Table 2, the GA with the surrogate model reaches on this function the same fitness values as the non-surrogate GA using only less than 30 per cent of the original fitness function evaluations (generation-based EC), or it is able to find 1.1-times better solution with 80 per cent of the original fitness evaluations (individual-based EC).

EC settings of the surrogate model	fitness of the best found individual	number of orig. fitness evaluations
without model	486.38 ± 56.5	4130 ± 1546
individual-based	544.73 ± 3.9	3241 ± 926
generation-based	490.28 ± 44.9	1185 ± 358

Table 2. GA performance without surrogate model and with the RBF/DSCL-based model; average results from 100 runs of the algorithm

5 Conclusion

Two kinds of surrogate models of expensive objective functions for mixed-variable continuous and discrete optimization were presented in this paper. Both of them make use of RBF networks; the first model focuses training of the RBF networks using clustering on the discrete part of the data while the second builds GLM on the discrete input variables. Detailed algorithms for training the models were provided. Results of testing on three different datasets showed that especially the second model is a competitive kind of regression for costly objective functions. Using the model on the benchmark fitness function resulted in saving up to 70 per cent of the original evaluations or 10 per cent increase of the final solution quality.

One of the most similar works dealing with surrogate models is the paper of Zhou [22]. He uses RBF networks as a local surrogate model in combination with a global model based on Gaussian processes. Other literature employs polynomials [8], Gaussian processes [4], or multilayer perceptron networks [10], but most publications consider only continuous or continuous and integer optimization.

References

1. L. Bajer, M. Holeňa: *Surrogate model for continuous and discrete genetic optimization based on RBF networks.* In C. Fyfe et al.,(Ed.), Intelligent Data Engineering and Automated Learning – IDEAL 2010, vol. 6283 of Lecture Notes in Computer Science, pp. 251–258. Springer, September 2010.
2. J.L. Bernier, J. Ortega, I. Rojas, E. Ros, A. Prieto: *Obtaining fault tolerant multilayer perceptrons using an explicit regularization.* Neural Processing Letters, 12(2), October 2000, 107–113.
3. A. J. Booker, J. Dennis, P.D. Frank, D.B. Serafini, V. Torczon, M. Trosset: *A rigorous framework for optimization by surrogates.* Structural and Multidisciplinary Optimization 17, 1999, 1–13.
4. D. Buche, N.N Schraudolph, P. Koumoutsakos: *Accelerating evolutionary algorithms with gaussian process fitness function models.* IEEE Trans. on Systems, Man, and Cybernetics, Part C: Applications and Reviews 35(2), 2005, 183–194.
5. M.D. Buhmann: *Radial basis functions: theory and implementations.* Cambridge Univ. Press, 2003.
6. D. Gorissen, I. Couckuyt, P. Demeester, T. Dhaene, K. Crombecq: *A surrogate modeling and adaptive sampling toolbox for computer based design.* Journal of Machine Learning Research 11, 2010, 2051–2055.
7. M. Holeňa, T. Cukic, U. Rodemerck, D. Linke: *Optimization of catalysts using specific, description-based genetic algorithms.* Journal of Chemical Information and Modeling 48(2), 2008, 274–282.
8. S. Hosder, L.T. Watson, B. Grossman: *Polynomial response surface approximations for the multidisciplinary design optimization of a high speed civil transport.* Optimization and Engineering 2(4), 2001, 431–452.
9. Y. Jin: *A comprehensive survey of fitness approximation in evolutionary computation.* Soft Computing – A Fusion of Foundations, Methodologies and Applications 9(1), 2005, 3–12.
10. Y. Jin, M. Hüsken, M. Olhofer, B. Sendhoff: *Neural networks for fitness approximation in evolutionary optimization.* Knowledge Incorporation in Evolutionary Computation, 2005, 281.
11. Q. Jun-Fei, Han Hong-Gui: *A repair algorithm for radial basis function neural network and its application to chemical oxygen demand modeling.* International Journal of Neural Systems 20(1), 2010, 63–74.
12. Cheol-Taek Kim, Ju-Jang Lee: *Training two-layered feedforward networks with variable projection method.* IEEE Transactions on Neural Networks 19(2), 2008, 371–375.
13. P. McCullagh, J.A. Nelder: *Generalized linear models.* Chapman & Hall, 1989.
14. S. Möhmel, N. Steinfeldt, S. Endgelschalt, M. Holeňa, S. Kolf, U. Dingerdissen, D. Wolf, R. Weber, M. Bewersdorf: *New catalytic materials for the high-temperature synthesis of hydrocyanic acid from methane and ammonia by high-throughput approach.* Applied Catalysis A: General 334, 2008, 73–83.
15. P. Narasimha, W.H. Delashmit, M.T. Manry, Jiang Li, F. Maldonado: *An integrated growing-pruning method for feedforward network training.* Neurocomputing 71(13-15), August 2008, 2831–2847.
16. M. Olhofer, T. Arima, T.S. B. Sendhoff, G. Japan: *Optimisation of a stator blade used in a transonic compressor cascade with evolution strategies.* Evolutionary Design and Manufacture, 45–54, 2000.
17. Y.S. Ong, P.B. Nair, A.J. Keane, K.W. Wong: *Surrogate-assisted evolutionary optimization frameworks for high-fidelity engineering design problems.* Knowledge Incorporation in Evolutionary Computation, 307–332, 2004.

18. L. Prechelt: *Proben1: A set of neural network benchmark problems and benchmarking rules.* Technical Report 21/94, 1994.
19. G. Singh, R. V. Grandhi: *Mixed-variable* optimization strategy employing multifidelity simulation and surrogate models. AIAA Journal 48(1), 2010, 215–223.
20. S. Valero, E. Argente, V. Botti: *DoE framework for catalyst development based on soft computing techniques.* Computers & Chem. Engineer. 33(1), 2009, 225–238.
21. A. Younis, Z. M. Dong: *Global optimization using mixed surrogate models for computation intensive designs.* In 2nd International Symposium on Computational Mechanics Hong Kong 1233, 2010, 1600–1605.
22. Z. Zhou, Y. S. Ong, P. B. Nair, A. J. Keane: *Combining global and local surrogate models to accelerate evolutionary optimization.* IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 37(1), 2007, 66–76.

Effective Datalog-like representation of procedural programs*

David Bednárek

Department of Software Engineering
Faculty of Mathematics and Physics, Charles University Prague
bednarek@ksi.mff.cuni.cz

Abstract. *Database systems are constantly extending their application area towards more general computing. However, applications that combine database access and general computing still suffer from the conceptual and technical gap between relational algebra and procedural programming. In this paper, we show that procedural programs may be effectively represented in a Datalog-like language with functions and aggregates. Such a language may then be used as a common representation for both relational and procedural part of an application.*

1 Introduction

As database systems were extending their application area, their mainstay language, SQL, became no longer sufficient to support all required applications of databases. The database community reacted in two ways: Implementing domain specific languages like XQuery, SPARQL etc. and improving the interaction of database engines with general procedural programming languages. The second approach is certainly more general but it is also more difficult due to deep differences between the procedural and relational paradigms.

Many database systems offer a procedural language with embedded SQL statements. The most common processing scenario is depicted at Fig. 1. The procedural and relational parts are separated already in the parser stage. The procedural part is converted to a procedural intermediate representation while the relational part is expressed using extended relational algebra. The two parts are processed almost independently in the following stages. When entering run time, the procedural part is usually expressed by a procedural bytecode which is designed for easy interpretation; the relational part is represented by a physical plan, i.e. an expression over a physical algebra.

Database-related optimizations are performed only on the relational side; individual SQL statements extracted from the source code are usually optimized independently. The most important optimization step is the strategy of physical plan generation, traditionally called *cost-based optimization.* Here, the logical relational algebra operators are converted to physical operators selected from a library and the selection is guided by database statistics and physical operator metadata (cost etc.).

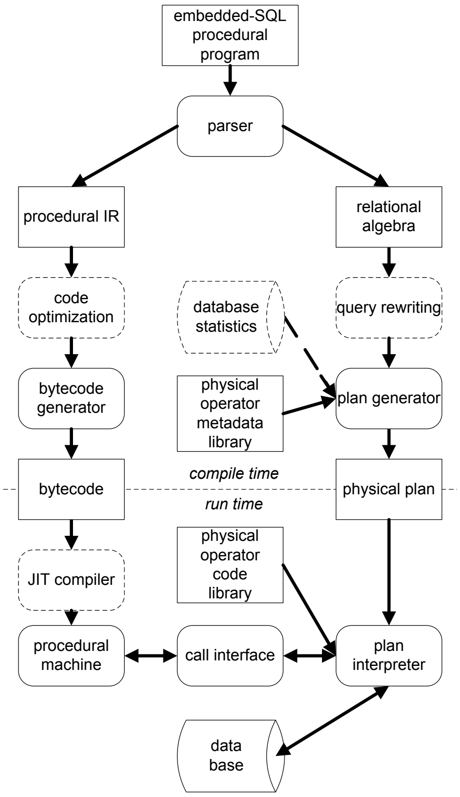


Fig. 1. Typical processing path of embedded-SQL programs

The procedural part meets with its relational elements only at run time – the procedural machine executes the procedural code containing calls to a database interface which in turn invokes the plan interpreter. The interpreter schedules and dispatches calls to procedures implementing individual physical operators of a plan. Although the individual SQL statements extracted from the source program often access the same data, their plans usually interact only at the lowest levels of physical data access; thus, they often repeat the same operations on the same data.

* The work was supported by the GACR project 202/10/0761 and by the GAUK grant SVV-2011-263312.

The most important reason for the separation of procedural and relational parts is probably the history – the relational (SQL) query engines were usually implemented well before the procedural *add-on* was considered and the software-engineering cost of reimplementing the query engine was considered too high.

Nevertheless, there is another reason for the separation – the nature of procedural code is so distant from the algebraic nature of SQL that it is very difficult to create a meaningful common representation for the two parts.

1.1 Goals

In this paper, we suggest using a Datalog-like language as the common intermediate representation for procedural and relational code. This idea is natural, considering the close relationship between Datalog and relational algebra on one side and the computing power of logic programming and recursion on the other side. However, there are still many obstacles in the integration effort. In particular, there is a risk of loss of efficiency since the procedural code is not evaluated directly but transformed to logic-programming representation and then evaluated by a procedural hardware.

In our approach, we strive to improve the efficiency of logic-programming representation by minimizing the size of the models – we show that a procedural code can be encoded in a logic program in such a manner that the size of the (minimal) model is proportional to the execution time of the original program on the same data. Thanks to this proportionality, the logic program may be evaluated completely in bottom-up manner.

Due to the focus on bottom-up evaluation, we prefer calling our approach *Datalog-like* over the term *logic programming*, although we certainly must use a language stronger than Datalog to achieve generality.

Besides function symbols which are necessary to simulate general procedural programming, our language needs negation and aggregation for the emulation of both procedural constructs and the embedded relational language. Both negation and aggregation require special handling to ensure well-defined semantics and there are several approaches to the semantics of negation in Datalog and logic programming in general.

Thus, defining a particular approach to semantics is a part of our effort – we reuse the concepts of *local stratification* [13] and *rule progressivity* [10] that together well reflect the nature of the original procedural code.

1.2 Architecture

The architecture of the proposed system is shown at Fig. 2. Instead of separating the procedural and relational part, the parser converts the source code into a Datalog-like intermediate representation. This intermediate language and the conversion of procedural code into it form the main subject of this paper. On the other hand, conversion of relational queries to Datalog is a well-known subject; thus, it is not necessary to describe the handling of embedded SQL statements.

The processing continues with rewriting step which optimizes the Datalog-like representation – this phase corresponds to query rewriting and it may also be influenced by database statistics. Moreover, several optimization algorithms known from compiler construction like loop unrolling or variable renaming [17] have their equivalents in the transformation of logic programs, so this phase covers also the optimization of procedural code.

The most important feature of this architecture is the ability to apply rewriting optimization step across the boundary between procedural and relational code. For instance, repeated invocations of a SQL statement may be glued together, offering, for instance, the possibility to cache their partial results.

The plan generator phase tries to cover the Datalog-like representation using a predefined set of patterns. Each pattern corresponds to a *component* which has several inputs and several outputs, each corresponding to a predicate. A simplest component cover one Datalog rule – in this case, the head of the rule corresponds to the output and each atom of the body corresponds to an input of the component. More sophisticated components correspond to a pattern covering more than one rule, including recursively dependent rules.

Some of the components correspond to physical operators of a relational engine; for instance, a Datalog rule with two body atoms may be implemented by a hash-join operator. Other components are implemented with simple procedural code snippets – these components ensure that the procedural parts of the source code are reverted back to procedural code. Of course, parts of the source code may be changed during the rewriting phase; consequently, procedural source code may be eventually covered by relational operators and, conversely, portions of the embedded relational statements may be converted to procedural snippets.

The implementation of physical relational operators is boxed in procedural packages which are connected together similarly to classical query plans. On the other hand, the procedural snippets are so small that individual packaging would be ineffective. Therefore, the snippets are combined together to larger procedural code fragments by the bytecode generator.

This paper deals with the design of the intermediate representation and the translation from procedural code to it. The subsequent steps – Datalog-based rewriting and plan generator – are subject of our current research. The run-time portion of the system at Fig. 2 was already implemented for a SPARQL compiler [4] whose compile time used a relational intermediate code and an algebra-based plan generator.

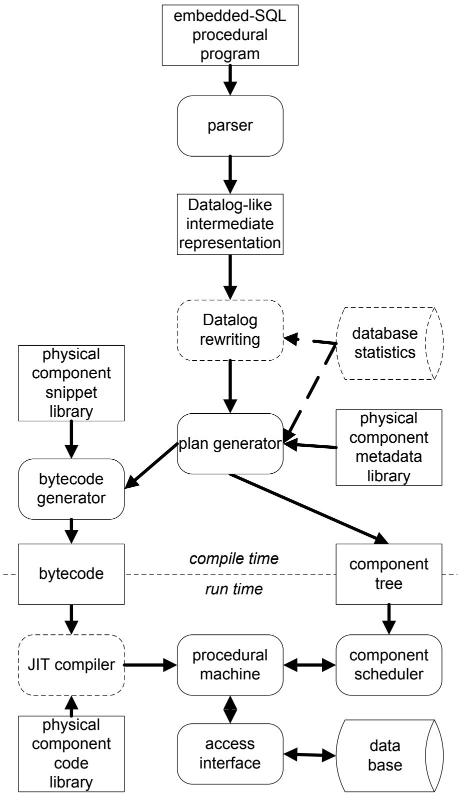


Fig. 2. Proposed processing path of embedded-SQL programs.

The rest of the paper is organized as follows: In the Section 2, we briefly review the related work on the interaction between procedural and relational code as well as Datalog-related definitions important for our approach. The following section uses a sample procedural code to illustrate several approaches to the modelling of procedural code in Datalog style. After reviewing the required extensions to Datalog in Section 3.5, a particular strategy to the minimization of model size is presented in Section 3.6.

2 Background and related work

Interaction of procedural and relational code was recognized as an important topic a long time ago; nevertheless, practical applications of the results are very scarce. In [7], a successful optimization of the interaction cost was shown in the case of calling procedurally-implemented functions from relational queries. Our approach also applies to this case; nevertheless, we focused on the opposite problem – repeated calling of relational queries from procedural code.

Nested relational algebra [14] may well reflect the use of structured and relation-valued variables in a procedural program; however, the overall computing strength of nested relational algebra is insufficient to express while loops. While loops may be added as an additional second-order construct atop of relational algebra, or represented by transitive closure in powerset algebra [9]. In Sec. 3.3, we will show a logical-programming equivalent of nested relational algebra together with the drawbacks of such an approach.

Flattening nested relations is an important step towards effective evaluation of nested algebra and it is also present in the core of our approach. The original flattening principle described in [16] was designed to flatten an isolated nested-relational algebra expression and it was based on the finite height of the expression tree. Since a while loop may generate a calculation of unlimited length, this flattening technique may not be applied for procedural programs. Instead, we had to use a numbering technique (see Sec. 3.4) and to solve some unwanted consequences of the numbering approach.

Datalog and its extensions, besides their natural applications in many areas of database theory, was already successfully used in areas related to procedural programming.

A language derived from logical programming was designed for programming in distributed environment [5]. The opposite problem, generating effective procedural implementations from Datalog programs, was studied in [11]. These recent publications suggest that the potential of Datalog was not exhausted in the first decades of its life and it may experience a revival fueled by the renewed interest in non-traditional database architectures.

There were attempts to improve the expressive power of Datalog towards procedural programming by non-traditional extensions of its semantics [8]. Extending Datalog towards complex data structures known from procedural programming was described in [6]. These powerful extensions relied on significant intrusions to the traditional Datalog semantics; consequently, their use in an intermediate language for a relational platform would be doubtful.

Datalog with temporal features was used to model sequences of data-manipulation statements [10] or in the analysis of procedural programs [15]. Our numbering technique shown in Sec. 3.4 is similar to temporal techniques although used for a different purpose.

Among the sheer number of approaches to Datalog extensions, semantics and evaluation strategies, *local* and, in particular, *temporal stratification* [12] has motivated our approach. In addition, we also make use of the *progressivity* [10] of rules in the scheduling strategy used in plan execution.

3 Modeling procedural code in Datalog style

Algorithm 1 Example: A procedural algorithm with embedded SQL statements

```
Require: Relation $M(A, B)$
 1: $S := z$
 2: while exists($M$) do
 3:     $X :=$ select min($A$) from $M$
            where $A$ not in (select $B$ from $M$)
 4:     $S := f(S, X)$
 5:     delete from $M$ where $A = X$
 6: end while
 7: return $S$
```

Algorithm 1 is an example of code written in a procedural language with SQL embedded. It consumes a relation M with schema (A, B) representing edges of a directed graph and traverses it in a topological ordering. In the loop, a node X is selected that has no incoming edge in M. The *min* aggregate is necessary to select from multiple candidates. Later in the loop, all outgoing edges of X are removed from M. The output of the loop is the variable S which aggregates the selected values of X using the constant z and the function $f(S, X)$ (e.g. concatenation). If the graph is acyclic, the algorithm terminates after at most $|M|$ iterations; otherwise, it eventually fails to find any A in the statement 3 and, depending on the subtle details of the statement semantics, either causes an exception or loops indefinitely.

Modeling of a while-loop requires an extension to relational algebra and transitive closure is the most obvious candidate. Each iteration of the loop changes the state of the program – the variables M and S – thus, there is a relation B which models the loop-body behavior using tuples $\langle M, S, M', S' \rangle$. Together with the while-head condition H, the transitive closure $L = \sigma_{M'=\emptyset}(\sigma_{M \neq \emptyset} B)^*$ models the loop. Unfortunately, it requires nested relational algebra to represent the relation-valued attribute M; although nested relational algebra can be simulated with plain relational algebra [16], this is not the case with transitive closure. Thus, representing the example code in the algebraic world requires transitive closure over nested relational algebra, which includes expensive atomic operations like equality test over sets and it is difficult to reduce it to implementable physical operators.

A natural response to the problems of algebraic representation is switching to Datalog where the while loop may be handled easily using recursion. However, the Algorithm 1 demonstrates several obstacles in the Datalog approach.

3.1 Naïve approach

The following rule forms a naïve Datalog implementation of the loop body:

$$\mathit{state}(M', S') \leftarrow \mathit{state}(M, S), \mathit{stmt}3(X, M),$$
$$\mathit{stmt}4(S', S, X), \mathit{stmt}5(M', M, X).$$

The predicates *stmt*3, *stmt*4, and *stmt*5 implement the behavior of the statements 3, 4, and 5 of Algorithm 1. This approach creates extremely inefficient representation because any model of this program contains ground atoms for *stmt*3, *stmt*4, and *stmt*5 representing all satisfiable variable assignments for the statements regardless of its reachability during an execution. In addition, statement clauses (not shown here) violate safety rules as some of the variables are bound only by functional symbols. Consequently, this approach is not suitable for bottom-up evaluation in Datalog style.

3.2 Using function symbols for relational algebra

The following, improved representation may be derived from the previous one using the Magic-sets transformation [1]:

$$\mathit{state}1(M) \leftarrow m0(M).$$
$$\mathit{state}2(M, z) \leftarrow \mathit{state}1(M).$$
$$\mathit{state}3(M, S) \leftarrow \mathit{state}2(M, S), M \neq \emptyset.$$
$$\mathit{state}3(M, S) \leftarrow \mathit{state}6(M, S), M \neq \emptyset.$$
$$\mathit{state}4(M, S, X) \leftarrow \mathit{state}3(M, S),$$
$$X = \pi_{min(A)}(\pi_A M \setminus \pi_B M).$$
$$\mathit{state}5(M, S', X) \leftarrow \mathit{state}4(M, S, X), S' = f(S, X).$$
$$\mathit{state}6(M', S) \leftarrow \mathit{state}5(M, S, X),$$
$$M' = M \setminus \sigma_{A=X} M.$$
$$\mathit{state}7(S) \leftarrow \mathit{state}2(M, S), M = \emptyset.$$
$$\mathit{state}7(S) \leftarrow \mathit{state}6(M, S), M = \emptyset.$$

The predicate state_i indicate the reachability of a particular variable assignment in the beginning of statement i of Algorithm 1. The relational statements are represented as equality statements containing function symbols from relational algebra. The rules implement a state machine simulating the execution of the original program; the model expansion and safety problems mentioned above disappeared.

The relational statements are implemented using a non-Datalog mechanism; thus, this approach is far from being a suitable intermediate representation for mixed code.

3.3 Nesting and unnesting relational variables

In this section, the relational algebra was hoisted to Datalog level using the *unnest* predicate and the *nest* aggregate. This approach is equivalent to nested relational algebra. The predicate $unnest(M, A, B)$ performs the membership test $\langle A, B\rangle \in M$; the generalized aggregate $nest(A, B)$ collects all $\langle A, B\rangle$ pairs and combine them into a set (see [3] for exact definiton of semantics of aggregates in Datalog):

$state1(M) \leftarrow m0(M).$
$state2(M, z) \leftarrow state1(M).$
$cond2(M) \leftarrow state2(M, S), unnest(M, A, B).$
$state3(M, S) \leftarrow state2(M, S), cond2(M).$
$state3(M, S) \leftarrow state6(M, S), cond2(M).$
$cond3(M, B) \leftarrow state3(M, S), unnest(M, A, B).$
$state4(M, S, min(A)) \leftarrow state3(M, S),$
  $unnest(M, A, B), \neg cond3(M, A).$
$state5(M, S', X) \leftarrow state4(M, S, X), S' = f(S, X).$
$state6(nest(A, B), S) \leftarrow state5(M, S, X),$
  $unnest(M, A, B), A \neq X.$
$state7(S) \leftarrow state2(M, S), \neg cond2(M).$
$state7(S) \leftarrow state6(M, S), \neg cond2(M).$

This approach unifies the means used for procedural and relational fragments. Nevertheless, it suffers from the stratification required by the *nest* aggregate and the need to incorporate all live variables into single $state_i$ atom.

3.4 Numbering iterations

The following code illustrates the approach we finally used. The argument T representing time (more exactly, the number of iteration) was introduced to almost all rules. It allowed dissolution of the original $state_i$ predicates: $state_i(T)$ indicates reachability of the statement i at time T.

$m2(1, A, B) \leftarrow m0(A, B).$
$s2(1, z).$
$state2(1).$
$state2(T + 1) \leftarrow branch23(T).$
$cond2(T) \leftarrow state2(T), m2(T, A, B).$
$branch23(T) \leftarrow state2(T), cond2(T).$
$cond3(T, B) \leftarrow branch23(T), m2(T, A, B).$
$x4(T, min(A)) \leftarrow branch23(T),$
  $m2(T, A, B), \neg cond3(T, A).$
$s2(T + 1, f(S, X)) \leftarrow branch23(T),$
  $s2(T, S), x4(T, X).$
$m2(T + 1, A, B) \leftarrow branch23(T),$
  $m2(T, A, B), x4(T, X), A \neq X.$
$branch27(T) \leftarrow state2(T), \neg cond2(T).$
$return(S) \leftarrow branch27(T), s2(T, S).$

Variable values are represented independently: $x_i(T, V)$ determines the value V of the variable X before entering statement i at time T. In the case of relational-valued variable M, the relation is unnested and its tuples are represented by individual instances of the atom $m_i(T, A, B)$.

Relational (and in general, complex-valued) Datalog variables and terms are no longer needed – every term is an atomic value.

Note that the dissolution of $state_i$ predicates created an opportunity for optimization: Every atomic statement modifies only one variable; therefore, only one clause is required for each statement, specifying the new variable value. For a variable, it is sufficient to have the value specified only at reference points (for S and M, it is the beginning of the statement 2), provided that at least one reference point lies on any path from any definition to any usage of this variable. In our case, the value for the reference point 2 is specified twice because there are two control paths leading to this point.

The execution of rules implementing statements is guarded by trigger predicates $branch_{i,j}$ which signalize passing from the statement i to the statement j. These predicates are controlled by conditions and their negations; in our case, the predicate *cond*2.

3.5 Requirements on the logic language

In our example Algorithm 1, there is a loop-carried dependence from the variable M through X to the next M value which involves negation and aggregation. This is reflected in the presence of negation and aggregation in the mutual recursion between the predicates $x4$ and $s2$. Our representation is therefore unstratifiable; the unstratifiability is inherent to Algorithm 1 since the length of the chain of negations generated by the loop is unlimited. Consequently, no stratification may exist for any Datalog-like representation of this example.

This forces us to use the concept of *local stratification*. Note that in pure Datalog without function symbols, local stratification is almost equivalent to stratification [2]. However, our system does use function symbols to generate the time values T and to implement built-in operators and functions of the procedural language and of SQL.

3.6 Long-range variable passing

In our example Algorithm 1, each iteration of the loop modifies each variable. However, a loop body may contain conditional statements; thus, a variable may become unmodified in some iterations. In this case, there must be a mechanism to pass the unmmodified variable value through the loop body. A simple implementation of such mechanism is the following clause:

$$value_V(T+1,X) \leftarrow value_V(T,X), cond(T).$$

For every scalar variable V and for every iteration T, a ground atom $value_V(T,X)$ is present in the model indicating the value X of the variable. Consequently, the model is of the size $\Omega(\tau v)$ where τ is the execution time of the procedural program and v is the number of variables in the program. For nonscalar variables the cost is even larger, because the argument X (or more arguments) encodes an individual element of a relation or an array, therefore there are as many ground atoms as the size of the variable. Evaluating the complete model is thus unacceptably costly.

Fortunately, the cost of unmodified variable passing may be lowered to $O(\tau\ log(v))$. The principle is depicted in Fig. 3 – instead of copying the variable value on every iteration, there are several layers of preferred points in time and copying is performed at these preferred points. Preferred points of layer k occur at the distance of 2^k iterations and copying is allowed either to a point of higher preference or to a point of access to the variable. The thick lines in Fig. 3 show how a variable is passed when it is accessed in the three points marked at the time axis.

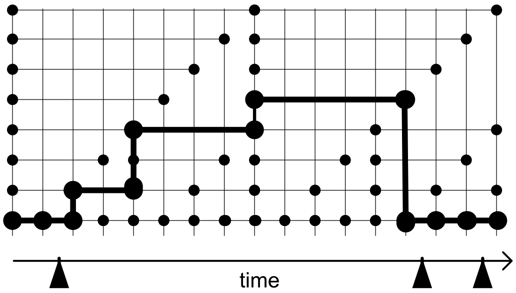


Fig. 3. Long-range variable passing

The number of copies done between two accesses is $O(log\Delta)$ where Δ is the time distance between the accesses, i.e. the length of the passing range. Since an atomic statement may access only a limited number of variables, the number n of passing ranges is proportional to the execution time, i.e. $n = c * \tau$. Since the ranges may not intersect for the same variable, their sum across of all variables is $\sum_{i=1}^{n} \Delta_i \leq \tau v$. Consequently, the total cost of copying is proportional to

$$\sum_{i=1}^{n} log(\Delta_i) \leq n\ log\frac{\sum_{i=1}^{n} \Delta_i}{n} \leq n\ log(\frac{\tau v}{n}) = \tau c\ log\frac{v}{c}$$

4 Conclusion

We have proposed a promising approach to mixed procedural and relational code whose key element is a novel intermediate representation based on logic programming. With respect to the assumed bottom-up evaluation strategy, the intermediate language falls to the Datalog family. Special care was taken for effective evaluation of the intermediate language – our results show only $O(log\ v)$ degradation with respect to the native procedural evaluation of a code with v variables.

In this paper, we presented the principles of the proposed language and its use for the representation of procedural code. For the sake of clarity, we omitted many technical details and tricks that were necessary to achieve the reported effectiveness of the representation.

Whether our approach is viable, it can be shown only by successful implementation of the whole processing chain from Fig. 2. The design of the intermediate representation was only a necessary prerequisite before attempting the implementation.

Our approach was motivated by the lessons learned from the implementation of a parallel SPARQL engine [4] for the Semantic Web. When using a relational repository, many Semantic Web algorithms are most easily expressed as simple procedural algorithms over relational queries.

Thus, using a combined relational/procedural intermediate language may save the tedious and errorprone work associated with reformulating such algorithms in either purely relational (if ever possible) or purely procedural way. In addition, the mixed representation offers new opportunities for optimization.

If successful, the new architecture may become important for areas where database access is tightly coupled with non-trivial computing, including the Semantic Web, computational linguistics or some areas of e-science.

References

1. C. Beeri, R. Ramakrishnan: *On the power of magic.* The Journal of Logic Programming, 10(3–4), 1991, 255–299.
2. H. A. Blair, V. W. Marek, J. S. Schlipf: *The expressiveness of locally stratified programs.* Annals of Mathematics and Artificial Intelligence, 15(2), 1995, 209–229.

3. M. P. Consens, A. O. Mendelzon: *Low-complexity aggregation in graphlog and datalog.* Theoretical Computer Science 116(1), 1993, 95–116.
4. Z. Falt, D. Bednarek, M. Cermak, F. Zavoral: *On parallel evaluation of sparql queries.* In DBKDA 2012, The Fourth International Conference on Advances in Databases, Knowledge, and Data Applications, 2012, 97–102.
5. S. C. Goldstein, F. Cruz: *Meld: A logical approach to distributed and parallel programming.* Technical report, DTIC Document, 2012.
6. S. Greco, L. Palopoli, E. Spadafora: *Extending datalog with arrays.* Data & Knowledge Engineering 17(1), 1995, 31–57.
7. R. Guravannavar, S. Sudarshan: *Rewriting procedures for batched bindings.* Proceedings of the VLDB Endowment 1(1), 2008, 1107–1123.
8. A. Guzzo, D. Sacca: *Semi-inflationary datalog: A declarative database language with procedural features.* Artificial Intelligence Communications 18(2), 2005, 79–92.
9. M. Gyssens, D. Van Gucht: *The powerset algebra as a result of adding programming constructs to the nested relational algebra.* ACM 17, 1988.
10. G. Lausen, B. Ludäscher, W. May: *On active deductive databases: The Statelog approach.* Transactions and Change in Logic Databases, 1998, 69–106.
11. Y. A. Liu, S. D. Stoller: *From datalog rules to efficient programs with time and space guarantees.* In Proceedings of the 5th ACM SIGPLAN International Conference on Principles and Practice of Declaritive Programming. ACM, 2003, 172–183.
12. C. Nomikos, P. Rondogiannis, M. Gergatsoulis: *Temporal stratification tests for linear and branching-time deductive databases.* Theoretical Computer Science 342(2), 2005, 382–415.
13. L. Palopoli: *Testing logic programs for local stratification.* Theoretical Computer Science 103(2), 1992, 205–234.
14. H. J. Schek, M. H. Scholl: *The relational model with relation-valued attributes.* Information Systems 11(2), 1986, 137–147.
15. Y. Smaragdakis, M. Bravenboer: *Using datalog for fast and easy program analysis.* In Datalog Reloaded: First International Workshop, Datalog 2010, Oxford, UK, March 16-19, 2010. Revised Selected Papers, vol. 6702, pp. 245. Springer-Verlag New York Inc, 2011.
16. J. Van den Bussche: *Simulation of the nested relational algebra by the flat relational algebra, with an application to the complexity of evaluating powerset algebra expressions.* Theoretical Computer Science 254(1-2), 2001, 363–377.
17. E. Visser: *Stratego: A language for program transformation based on rewriting strategies system description of stratego 0.5.* In Rewriting Techniques and Applications, Springer, 2001, 357–361.

Conformal sets in neural network regression*

Radim Demut[1] and Martin Holeňa[2]

[1] Faculty of Nuclear Sciences and Physical Engineering
Czech Technical University in Prague
demut@seznam.cz
[2] Institute of Computer Science
Academy of Sciences of the Czech Republic
martin@cs.cas.cz

Abstract. *This paper is concerned with predictive regions in regression models, especially neural networks. We use the concept of conformal prediction (CP) to construct regions which satisfy given confidence level. Conformal prediction outputs regions, which are automatically valid, but their width and therefore usefulness depends on the used nonconformity measure. A nonconformity measure should tell us how different a given example is with respect to other examples. We define nonconformity measures based on some reliability estimates such as variance of a bagged model or local modeling of prediction error. We also present results of testing CP based on different nonconformity measures showing their usefulness and comparing them to traditional confidence intervals.*

1 Introduction

This paper is concerned with predictive regions for regression models, especially neural networks. We often want to know not only the label y of a new object, but also how accurate the prediction is. Could the real label be very far from our prediction or is our prediction very accurate? It is possible to use traditional confidence intervals to answer this question but they do not work very well with highly nonlinear regression models such as neural networks. We use conformal prediction to solve this problem and construct some accurate and useful prediction regions.

We introduce conformal prediction (CP) in chapter 2. Conformal prediction does not output single label but a set of labels Γ^ε. The size of the prediction set depends on a significance level ε which we want to achieve. Significance level is under some conditions the probability that our prediction lies outside the set. The set is smaller for larger ε. If we have some prediction rule, we will call it simple predictor and we can use it to construct conformal predictor. We introduce transductive conformal predictors where the prediction rule is updated after a new example arrives. But these predictors are not suitable for neural network regression, therefore, we also introduce inductive conformal predictors where the prediction rule is updated only after a given number of new examples has arrived and a calibration set is used.

In order to define a conformal predictor we need a suitable nonconformity measure. A nonconformity measure should tell us how different a given example is with respect to other examples. In chapter 3, we introduce two reliability estimates: variance of a bagged model and local modeling of prediction error. We use these reliability estimates in chapter 4 to define normalized nonconformity measures. Some other reliability estimates could be used, e.g. sensitivity analysis or density based reliability estimate.

In chapter 5, we use CP, based on nonconformity measures defined in chapter 4, on testing data to compare our conformal regions with traditional confidence intervals and with conformal intervals where these traditional confidence intervals are used to construct the nonconformity measure.

2 Conformal prediction

We assume that we have an infinite sequence of pairs

$$(x_1, y_1), (x_2, y_2), \ldots, \tag{1}$$

called *examples*. Each example (x_i, y_i) consists of an *object* x_i and its *label* y_i. The objects are elements of a measurable space $\mathbf{X}$ called the *object space* and the labels are elements of a measurable space $\mathbf{Y}$ called the *label space*. Moreover, we assume that $\mathbf{X}$ is non-empty and that the σ-algebra on $\mathbf{Y}$ is different from $\{\emptyset, \mathbf{Y}\}$. We denote $z_i := (x_i, y_i)$ and we set

$$\mathbf{Z} := \mathbf{X} \times \mathbf{Y} \tag{2}$$

and call $\mathbf{Z}$ the *example space*. Thus the infinite data sequence (**??**) is an element of the measurable space $\mathbf{Z}^\infty$.

Our standard assumption is that the examples are chosen independently from some probability distribution Q on $\mathbf{Z}$, i.e. the infinite data sequence (**??**) is drawn from the power probability distribution Q^∞ on

* This work was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS12/157/OHK4/2T/14 and the Czech Science Foundation grant 201/08/0802.

$\mathbf{Z}^\infty$. Usually we need only slightly weaker assumption that the infinite data sequence (**??**) is drawn from a distribution P on $\mathbf{Z}^\infty$ that is exchangeable, that means that every $n \in \mathbb{N}$, every permutation π of $\{1, \ldots, n\}$, and every measurable set $E \subseteq \mathbf{Z}^\infty$ fulfill

$$P\{(z_1, z_2, \ldots) \in \mathbf{Z}^\infty : (z_1, \ldots, z_n) \in E\} = \\ P\{(z_1, z_2, \ldots) \in \mathbf{Z}^\infty : (z_{\pi(1)}, \ldots, z_{\pi(n)}) \in E\}$$

We denote $\mathbf{Z}^*$ the set of all finite sequences of elements of $\mathbf{Z}$, $\mathbf{Z}^n$ the set of all sequences of elements of $\mathbf{Z}$ of length n. The order in which old examples appear should not make any difference. In order to formalize this point we need the concept of a bag. A *bag* of size $n \in \mathbb{N}$ is a collection of n elements some of which may be identical. To identify a bag we must say what elements it contains and how many times each of these elements is repeated. We write $\backslash z_1, \ldots, z_n /$ for the bag consisting of elements $z_1, \ldots, z_n$, some of which may be identical with each other. We write $\mathbf{Z}^{(n)}$ for the set of all bags of size n of elements of a measurable space $\mathbf{Z}$. We write $\mathbf{Z}^{(*)}$ for the set of all bags of elements of $\mathbf{Z}$.

2.1 Confidence predictors

We assume that at the nth trial we have firstly only the object x_n and only later we get the label y_n. If we want to predict y_n, we need a *simple predictor*

$$D : \mathbf{Z}^* \times \mathbf{X} \to \mathbf{Y} \ . \tag{3}$$

For any sequence of old examples $x_1, y_1, \ldots, x_{n-1}, y_{n-1} \in \mathbf{Z}^*$ and any new object x_n, it gives $D(x_1, y_1, \ldots, x_{n-1}, y_{n-1}, x_n) \in \mathbf{Y}$ as its prediction for the new label y_n.

Instead of merely choosing a single element of $\mathbf{Y}$ as our prediction for y_n, we want to give subsets of $\mathbf{Y}$ large enough that we can be confident that y_n will fall in them, while also giving smaller subsets in which we are less confident. An algorithm that predicts in this sense requires additional input $\varepsilon \in (0, 1)$, which we call significance level, the complementary value $1 - \varepsilon$ is called confidence level. Given all these inputs

$$x_1, y_1, \ldots, x_{n-1}, y_{n-1}, x_n, \varepsilon \tag{4}$$

an algorithm Γ that interests us outputs a subset

$$\Gamma^\varepsilon(x_1, y_1, \ldots, x_{n-1}, y_{n-1}, x_n) \tag{5}$$

of $\mathbf{Y}$. We require this subset to shrink as ε is increased that means it holds

$$\Gamma^{\varepsilon_1}(x_1, y_1, \ldots, x_{n-1}, y_{n-1}, x_n) \subseteq \\ \Gamma^{\varepsilon_2}(x_1, y_1, \ldots, x_{n-1}, y_{n-1}, x_n) \tag{6}$$

whenever $\varepsilon_1 \geq \varepsilon_2$.

Formally, a *confidence predictor* is a measurable function

$$\Gamma : \mathbf{Z}^* \times \mathbf{X} \times (0, 1) \to 2^{\mathbf{Y}} \tag{7}$$

that satisfies (**??**) for all $n \in \mathbb{N}$, all incomplete data sequences $x_1, y_1, \ldots, x_{n-1}, y_{n-1}, x_n$ and all significance levels $\varepsilon_1 \geq \varepsilon_2$.

Whether Γ makes an error on the nth trial of the data sequence $\omega = (x_1, y_1, x_2, y_2, \ldots)$ at significance level ε can be represented by a number that is one in case of an error and zero in case of no error

$$\mathrm{err}_n^\varepsilon(\Gamma, \omega) := \begin{cases} 1 \text{ if } y_n \notin \Gamma^\varepsilon(x_1, y_1, \ldots, \\ \quad x_{n-1}, y_{n-1}, x_n) \ , \\ 0 \text{ otherwise } \ , \end{cases} \tag{8}$$

and the number of errors during the first n trials is

$$\mathrm{Err}_n^\varepsilon(\Gamma, \omega) := \sum_{i=1}^{n} \mathrm{err}_i^\varepsilon(\Gamma, \omega) \ . \tag{9}$$

If ω is drawn from an exchangeable probability distribution P, the number $\mathrm{err}_n^\varepsilon(\Gamma, \omega)$ is the realized value of a random variable, which we may designate $\mathrm{err}_n^\varepsilon(\Gamma, P)$. We say that confidence predictor Γ is *conservatively valid* if for any exchangeable probability distribution P on $\mathbf{Z}^\infty$ there exist two families

$$(\xi_n^{(\varepsilon)} : \varepsilon \in (0, 1), n = 1, 2, \ldots) \tag{10}$$

and

$$(\eta_n^{(\varepsilon)} : \varepsilon \in (0, 1), n = 1, 2, \ldots) \tag{11}$$

of $\{0, 1\}$-valued variables such that

- for a fixed ε, $\xi_1^{(\varepsilon)}, \xi_2^{(\varepsilon)}, \ldots$ is a sequence of independent Bernoulli random variables with parameter ε;
- for all n and ε, $\eta_n^{(\varepsilon)} \leq \xi_n^{(\varepsilon)}$;
- the joint distribution of $\mathrm{err}_n^\varepsilon(\Gamma, P)$, $\varepsilon \in (0, 1)$, $n = 1, 2, \ldots$, coincides with the joint distribution of $\eta_n^{(\varepsilon)}$, $\varepsilon \in (0, 1)$, $n = 1, 2, \ldots$.

2.2 Transductive conformal predictors

A *nonconformity measure* is a measurable mapping

$$A : \mathbf{Z}^{(*)} \times \mathbf{Z} \to \overline{\mathbb{R}} \ . \tag{12}$$

To each possible bag of old examples and each possible new example, A assigns a numerical score indicating how different the new example is from the old ones. It is sometimes convenient to consider separately how a nonconformity measure deals with bags of different sizes. If A is a nonconformity measure, for each $n = 1, 2, \ldots$ we define a function

$$A_n : \mathbf{Z}^{(n-1)} \times \mathbf{Z} \to \overline{\mathbb{R}} \tag{13}$$

as the restriction of A to $\mathbf{Z}^{(n-1)} \times \mathbf{Z}$. The sequence $(A_n : n \in \mathrm{I\!N})$, which we abbreviate to (A_n) will also be called a *nonconformity measure.*

Given a nonconformity measure (A_n) and a bag $\backslash z_1, \ldots, z_n /$ we can compute the nonconformity score

$$\alpha_i := A_n(\backslash z_1, \ldots, z_{i-1}, z_{i+1}, \ldots z_n /, z_i) \qquad (14)$$

for each example z_i in the bag. Because a nonconformity measure (A_n) may be scaled however we like, the numerical value of α_i does not, by itself, tell us how unusual (A_n) finds z_i to be. For that we define *p-value* for z_i as

$$p := \frac{|\{j = 1, \ldots, n : \alpha_j \geq \alpha_i\}|}{n} \ . \qquad (15)$$

We define *transductive conformal predictor* (TCP) by a nonconformity measure (A_n) as a confidence predictor $\mathit{\Gamma}$ obtained by setting

$$\mathit{\Gamma}^{\varepsilon}(x_1, y_1, \ldots, x_{n-1}, y_{n-1}, x_n) \qquad (16)$$

equal to the set of all labels $y \in \mathbf{Y}$ such that

$$\frac{|\{i = 1, \ldots, n : \alpha_i(y) \geq \alpha_n(y)\}|}{n} > \varepsilon \ , \qquad (17)$$

where

$$\begin{aligned}\alpha_i(y) := A_n(&\backslash(x_1, y_1), \ldots, (x_{i-1}, y_{i-1}), \\ &(x_{i+1}, y_{i+1}), \ldots, (x_{n-1}, y_{n-1}), (x_n, y)/, \\ &(x_i, y_i)) \ , \quad \forall i = 1, \ldots, n-1 \ , \\ \alpha_n(y) := A_n(&\backslash(x_1, y_1), \ldots, (x_{n-1}, y_{n-1})/, (x_n, y)) \ .\end{aligned}$$

We now remind an important property of TCP. The proof of the following theorem can be found in [?].

Theorem 1. *All conformal predictors are conservatively valid.*

If we are given a simple predictor (**??**) whose output does not depend on the order in which the old examples are presented, than the simple predictor D defines a prediction rule $D_{\backslash z_1, \ldots, z_n /} : \mathbf{X} \to \mathbf{Y}$ by the formula

$$D_{\backslash z_1, \ldots, z_n /}(x) := D(z_1, \ldots, z_n, x) \ . \qquad (18)$$

A natural measure of nonconformity of z_i is the deviation of the predicted label

$$\widehat{y}_i := D_{\backslash z_1, \ldots, z_n /}(x_i) \qquad (19)$$

from the true label y_i. We can also use the deleted prediction defined as

$$\widehat{y}_{(i)} := D_{\backslash z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_n /}(x_i) \ . \qquad (20)$$

A *discrepancy measure* is a measurable function

$$\mathit{\Delta} : \mathbf{Y} \times \mathbf{Y} \to \mathrm{I\!R} \ . \qquad (21)$$

Given a simple predictor D and a discrepancy measure $\mathit{\Delta}$ we define functions (A_n) as follows: for any $((x_1, y_1), \ldots, (x_n, y_n)) \in \mathbf{Z}^*$, the values

$$\begin{aligned}\alpha_i = A_n(&\backslash(x_1, y_1), \ldots, (x_{i-1}, y_{i-1}), \\ &(x_{i+1}, y_{i+1}), \ldots, (x_n, y_n)/, (x_i, y_i))\end{aligned} \qquad (22)$$

are defined according to (**??**) and (**??**) by the formula

$$\alpha_i := \mathit{\Delta}(y_i, D_{\backslash z_1, \ldots, z_n /}(x_i)) \qquad (23)$$

and the formula

$$\alpha_i := \mathit{\Delta}(y_i, D_{\backslash z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_n /}(x_i)) \ , \qquad (24)$$

respectively. It can be easily checked that in both cases (A_n) form a nonconformity measure.

2.3 Inductive conformal predictors

In TCP, we need to compute the p-value (**??**) for all labels $y \in \mathbf{Y}$ to determine the set $\mathit{\Gamma}^{\varepsilon}$. In the case of regression, we have $\mathbf{Y} = \mathrm{I\!R}$ and it is not possible to try each $y \in \mathbf{Y}$. Sometimes it is possible to generally solve equations $\alpha_i(y) \geq \alpha_n(y)$ with respect to y, and therefore determine the set $\mathit{\Gamma}^{\varepsilon}$. But if we use neural networks as simple predictor, we do not know the general form of the simple predictor, i.e. we do not know a functional relationship between the training set and the trained network, because random influences enter the training algorithm. Hence, we cannot solve the equations $\alpha_i(y) \geq \alpha_n(y)$, and it is not possible to use TCP. Even if the equations can be solved, it can be very computationally inefficient.

To avoid this problem we can use *inductive conformal predictor* (ICP). To define ICP from a nonconformity measure (A_n) we fix a finite or infinite increasing sequence of positive integers $m_1, m_2, \ldots$ (called update trials). If the sequence is finite we add one more member equal to infinity at the end of the sequence. We need more than m_1 training examples. Then we find k such that $m_k < n \leq m_{k+1}$. The ICP is determined by (A_n) and the sequence $m_1, m_2, \ldots$ of update trials is defined to be the confidence predictor $\mathit{\Gamma}$ such that the prediction set

$$\mathit{\Gamma}^{\varepsilon}(x_1, y_1, \ldots, x_{n-1}, y_{n-1}, x_n) \qquad (25)$$

is equal to the set of all labels $y \in \mathbf{Y}$ such that

$$\frac{|\{j = m_k + 1, \ldots, n : \alpha_j \geq \alpha_n(y)\}|}{n - m_k} > \varepsilon \ , \qquad (26)$$

where the nonconformity scores are defined by

$$\alpha_j := A_{m_k+1}(\backslash(x_1,y_1),\ldots,(x_{m_k},y_{m_k})/,(x_j,y_j)) \ , \quad \text{for } j = m_k+1,\ldots,n-1 \qquad (27)$$

$$\alpha_n := A_{m_k+1}(\backslash(x_1,y_1),\ldots,(x_{m_k},y_{m_k})/, (x_n,y)) \ . \qquad (28)$$

The proof of the following theorem can be found in [**?**].

Theorem 2. *All ICPs are conservatively valid.*

For ICP combining (**??**) with (**??**) and (**??**) we get

$$A_{l+1}(\ \backslash(x_1,y_1),\ldots,(x_l,y_l)/,(x,y)) = \Delta(y, D_{\backslash(x_1,y_1),\ldots,(x_l,y_l),(x,y)/}(x)) \qquad (29)$$

and

$$A_{l+1}(\ \backslash(x_1,y_1),\ldots,(x_l,y_l)/,(x,y)) = \Delta(y, D_{\backslash(x_1,y_1),\ldots,(x_l,y_l)/}(x)) \ , \qquad (30)$$

respectively. When we define A by (**??**), we can see that the ICP requires recomputing the prediction rule only at the update trials $m_1, m_2, \ldots$. We will use the simplest case, where there is only one update trial m_1, therefore, we compute the prediction rule only once.

3 Reliability estimates

In this chapter we are interested in different approaches to estimate the reliability of individual predictions in regression.

3.1 Variance of a bagged model

We are given a learning set $L = \{(x_1,y_1),\ldots,(x_n,y_n)\}$ and take repeated bootstrap samples $L^{(i)}$, $i = 1,\ldots,m$ of size d from the learning set, i.e. for $i = 1,\ldots,m$ we randomly choose d points from the original learning set L with the return and put them in $L^{(i)}$. The number of points d can be chosen arbitrary. We induce a new model on each of these bootstrap samples $L^{(i)}$. Each of the models yields a prediction $K_i(x)$, $i = 1,\ldots,m$ for a considered input x. The label of the example x is predicted by averaging the individual predictions

$$K(x) := \frac{\sum_{i=1}^{m} K_i(x)}{m} \ . \qquad (31)$$

We call this procedure *bootstrap aggregating* or *bagging*. The reliability estimate of a bagged model is defined as the prediction variance

$$\mathrm{BAGV}(x) := \frac{1}{m}\sum_{i=1}^{m}(K_i(x) - K(x))^2 \ . \qquad (32)$$

3.2 Local modeling of prediction error

We find k nearest neighbors of an unlabeled example x in the training set, therefore, we have a set $N = \{(x_1,y_1),\ldots,(x_k,y_k)\}$ of nearest neighbors. We define the estimate denoted CNK for an unlabeled example x as the difference between the average label of the nearest neighbors and the example's prediction y (using the model that was generated on all learning examples)

$$\mathrm{CNK}(x) := \frac{\sum_{i=1}^{k} y_i}{k} - y \ . \qquad (33)$$

The dependence on x on the right hand side of the previous equation is implicit, but both the prediction y and the selection of nearest neighbors depends on x.

4 Normalized nonconformity measures

We will follow a similar approach as is used in the article [**?**], but we will incorporate the reliability estimates from previous chapter and use it for neural network regression.

We will use ICP with only one update trial. Let us have training set of size l, where $l > m_1$. We will split it into two sets, the proper training set T of size m_1 (we will further write m) and the calibration set C of size $q = l - m$. We will use the proper training set for creating the simple predictor $D_{\backslash(x_1,y_1),\ldots,(x_m,y_m)/}$. The calibration set is used for calculating the p-value of new test examples. It is good to first normalize the data (i.e. subtract the mean and divide data by sample variance).

We will denote r_i any of the previously defined reliability estimates in the point x_i with given simple predictor D. We compute r_i for all points in the calibration set and define R_i for any given point x_i as

$$R_i := \frac{r_i}{\mathrm{median}\{r_j : r_j \in C\}} \ . \qquad (34)$$

We define a discrepancy measure (**??**) as

$$\Delta(y_1,y_2) := \left|\frac{y_1 - y_2}{\gamma + R_i}\right| \ , \qquad (35)$$

where parameter $\gamma \geq 0$ controls the sensitivity to changes of R_i. Then, we get the nonconformity score

$$\alpha_i(y) = \left|\frac{y - \hat{y}_i}{\gamma + R_i}\right| \ . \qquad (36)$$

We sort nonconformity scores of the calibration examples in descending order

$$\alpha_{(m+1)} \geq \cdots \geq \alpha_{(m+q)} \ , \qquad (37)$$

and denote

$$s = \lfloor \varepsilon(q+1) \rfloor \ . \qquad (38)$$

Proposition 1. *The prediction set Γ^ε of the new test example x_{l+g} (where x_{l+g} is from the infinite sequence (??)) given the nonconformity score (??) is equal to the interval*

$$\langle \hat{y}_{l+g} - \alpha_{(m+s)}(\gamma + R_{l+g}), \hat{y}_{l+g} + \alpha_{(m+s)}(\gamma + R_{l+g}) \rangle \ . \qquad (39)$$

Proof. To compute the prediction set Γ^ε of the new test example x_{l+g} we need to find all $y \in \mathbf{Y}$ such that for the p-value it holds

$$\begin{aligned} & p(y) = \\ & \frac{|\{i = m+1, \dots, m+q, l+g : \alpha_i \geq \alpha_{l+g}(y)\}|}{q+1} \\ & > \varepsilon \ . \end{aligned} \qquad (40)$$

We multiply the inequality by $q + 1$ and then it is equivalent to

$$|\{i = m+1, \dots, m+q, l+g : \alpha_i \geq \alpha_{l+g}(y)\}| > \lfloor \varepsilon(q+1) \rfloor \qquad (41)$$

and this inequality holds if and only if

$$\alpha_{(m+s)} \geq \alpha_{l+g}(y) = \left| \frac{y - \hat{y}_{l+g}}{\gamma + R_{l+g}} \right| \ . \qquad (42)$$

From (??) follows the assertion of the proposition.

5 Simulation

We carried out a simulation to test the normalized nonconformity measures based on different reliability estimates. We used neural networks with radial basis functions (RBF networks) as our regression models with Gaussian used as the basis function. Therefore, the output of the RBF network $f : \mathbb{R}^n \to \mathbb{R}$ has the form

$$f(\mathbf{x}) = \sum_{i=1}^{N} \pi_i \exp\left\{-\beta_i ||\mathbf{x} - \mathbf{c_i}||^2\right\} \ , \qquad (43)$$

where N is the number of neurons in the hidden layer, $\mathbf{c_i}$ is the center vector for neuron i, β_i determines the width of the ith neuron and π_i are the weights of the linear output neuron. RBF networks are universal approximators on a compact subset of $\mathbb{R}^n$. This means that a RBF network with enough hidden neurons can approximate any continuous function with arbitrary precision.

We used a benchmark function similar to some empirical functions encountered in chemistry to carry out our experiment. This function was introduced in [?]. The value of this function ϑ in the point $(x_1, x_2, x_3, x_4, x_5)$ can be expressed as

$$\begin{aligned} \vartheta(x_1, x_2, x_3, x_4, x_5) &= -A(x_1, x_2) \\ & -B(x_2, x_3)C(x_3, x_4, x_5) \ , \end{aligned} \qquad (44)$$

where

$$\begin{aligned} A(x_1, x_2) &= 0.6g(x_1 - 0.35, x_2 - 0.35) \\ &\quad +0.75g(x_1 - 0.1, x_2 - 0.1) \\ &\quad +g(x_1 - 0.35, x_2 - 0.1) \\ B(x_2, x_3) &= 0.4g(x_2 - 0.1, x_3 - 0.3) \\ C(x_3, x_4, x_5) &= 5 + 25[1 - \{1 + (x_3 - 0.3)^2 \\ &\quad +(x_4 - 0.15)^2 + (x_5 - 0.1)^2\}^{1/2}] \\ g(a, b) &= 100 - \sqrt{(100a)^2 + (100b)^2} \\ &\quad +50\frac{\sin\sqrt{(100a)^2 + (100b)^2}}{\sqrt{(100a)^2 + (100b)^2 + (0.01)^2}} \ . \end{aligned}$$

Moreover, the input vectors must satisfy following conditions

$$\sum_{i=1}^{5} x_i = 1 \quad \text{and} \quad x_i \in [0, 1], \text{for } i = 1, \dots, 5 \ . \qquad (45)$$

We repeated the following procedure five times for region with significance level 0.1 and five times for region with significance level 0.05.

- Randomly generate 600 points satisfying the conditions (??).
- Compute the function values of function ϑ in these points.
- Normalize data (i.e. subtract the mean and divide data by sample variance)
- Split this set of points into a training set of 500 points and a testing set of 100 points.
- Split the training set into a proper training set of 401 points and a calibration set of 99 points (then, we divide the p-value in (??) by 100).
- Split the proper training set on training set for fitting the RBF network and the validation set. Fit the RBF network with 1, 2, 3, 4 and 5 hidden neurons ten times using the Matlab function lsqcurvefit.
- Choose the RBF network with the smallest error on the validation set for each number of hidden neurons.
- Compute the prediction sets for each of the 100 testing points for each number of hidden neurons.
- Transform data and predictive regions back to the original size (i.e. multiply by the original sample variance and add the original mean)
- Determine if the original point lies in our prediction sets.

The initial values of parameters π_i were set as mean of the response vector, initial values of β_i were set as the mean of the standard deviation of the components of training data points. The centers $\mathbf{c_i}$ were set randomly.

We also computed confidence intervals using Matlab function nlpredci (denoted Conf Int). The Jacobian can be computed exactly, because the form of the RBF network is known and differentiable. Therefore, we supply the function nlpredci with this Jacobian. We also use the width of this interval as another reliability estimate for our normalized nonconformity measure.

We compare normalized nonconformity measures based on the following reliability estimates: the local modeling of prediction errors using nearest neighbors (CNK), the variance of a bagged model (BAGV) and the width of confidence intervals (CONF).

The variance of a bagged model was computed for number of different models $m = 10$ and the bootstrap samples were as big as the original sample.

The CNK estimates were computed for number of neighbors $k = 2, 5, 10$.

We present the results of testing CP based on different nonconformity measures in Figures **??**, **??** and **??**. There is a boxplot of all labels in Figure **??** to compare the range of all labels with the width of different predictive regions. Figures **??** and **??** show boxplots of the width of prediction regions for significance levels $\varepsilon = 0.1$ and $\varepsilon = 0.05$, respectively. It is not only interesting whether the intervals are small enough, but they should also be valid. The percentage of labels inside the predictive regions are in Tables **??** and **??** for significance levels $\varepsilon = 0.1$ and $\varepsilon = 0.05$, respectively.

The results for traditional confidence intervals computed by Matlab function nlpredci are not shown in the figures, because these results are very different from the others. The median width for these intervals lies between 10^{10} and 10^{14} for all counts of neurons. This is probably because of the highly nonlinear character of neural nets, while nlpredci is based on linearization. Moreover, during the computation of these intervals a Jacobian matrix must be inverted but this matrix was very often ill conditioned, therefore, the results for confidence intervals are not too reliable.

Despite what was said in the previous paragraph, the predictive regions based on the width of confidence intervals produce sensible results. But these prediction regions show highest inconsistency between different neuron counts and have highest number of very large intervals. These regions produce sometimes very good results, but they are probably very dependent on the actual fit of the neural network and their results are not as consistent as the results of the other methods. However, we can see in Tables **??** and **??** that these intervals are valid as the percentage of labels inside predictive regions is always slightly higher than the confidence level.

Results for predictive regions based on the local modeling of prediction errors depend a little bit on the count of nearest neighbors. These intervals are valid for all numbers of neighbors, but the tightest intervals were achieved for two neighbors. The difference between using five or ten neighbors is not too big but lower number of neighbors works better in our model. This is probably caused by our data and it seems that only a few neighbors are relevant to our prediction. These regions are also the easiest and fastest to compute.

The best results among all predictive regions are achieved by those based on a variance of a bagged model. These regions are the tightest of all tested and they do not vary as much as those based on confidence intervals. These regions also maintain the validity. The drawback of these regions is that we need to fit a lot of additional models which takes a lot of time in the case of neural network regression. But if time and computational efficiency is not a problem then this method produces best regions.

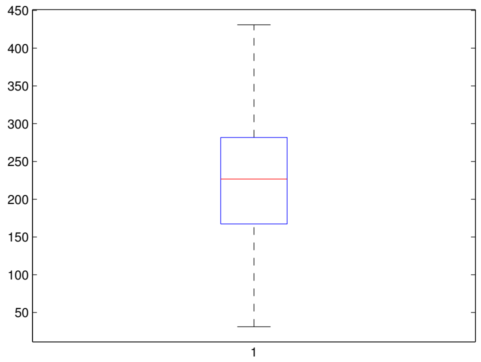


Fig. 1. Boxplot of all labels.

Neurons	CNK2	CNK5	CNK10	BAGV	CONF
2	91.0	91.2	90.0	91.4	92.4
3	92.6	92.4	92.6	94.0	93.6
4	92.2	90.4	90.0	90.4	90.2
5	94.6	92.6	90.0	90.2	90.8
6	92.8	89.8	91.8	91.6	91.8

Table 1. Percentage of labels inside predictive regions for $\varepsilon = 0.1$.

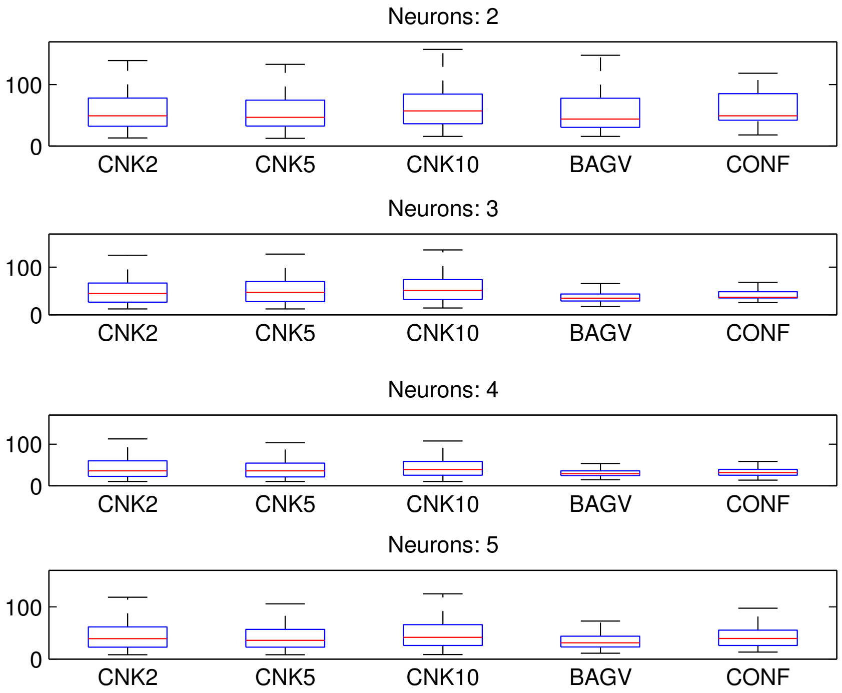


Fig. 2. Interval widths for $\varepsilon = 0.1$.

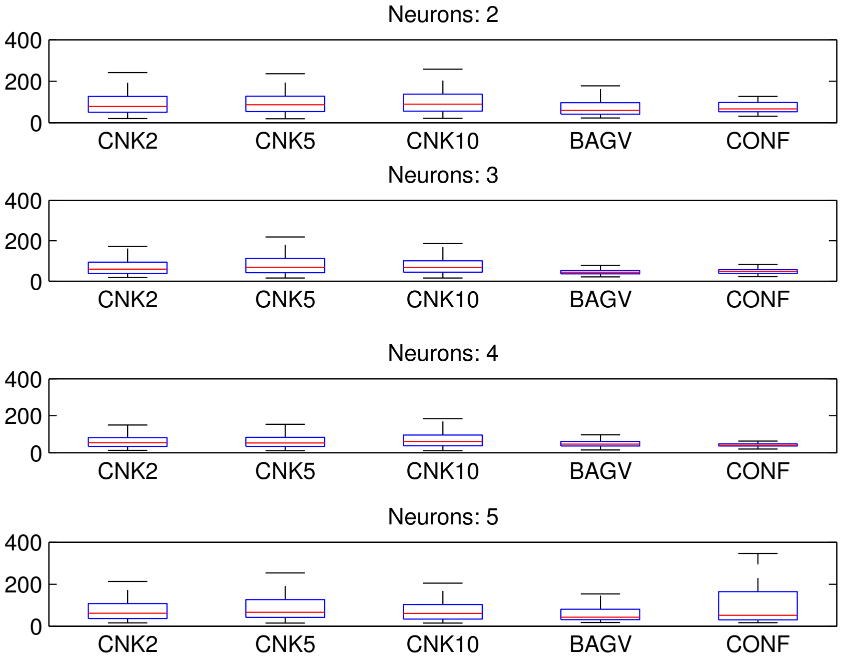


Fig. 3. Interval widths for $\varepsilon = 0.05$.

Neurons	CNK2	CNK5	CNK10	BAGV	CONF
2	95.8	96.8	96.8	96.2	95.6
3	97.2	96.8	96.8	97.8	95.6
4	96.2	96.4	96.6	97.4	97.0
5	97.2	97.6	96.4	96.4	97.6
6	97.2	98.0	96.4	97.4	96.8

Table 2. Percentage of labels inside predictive regions for $\varepsilon = 0.05$.

6 Conclusion

We presented several methods for computing predictive regions in neural network regressions. These methods are based on the inductive conformal prediction with novel nonconformity measures proposed in this paper. Those measures use reliability estimates to determine how different a given example is with respect to other examples. We compared our new predictive regions with traditional confidence intervals on testing data. The confidence intervals did not perform very well, the intervals were too large, it was probably caused by the high nonlinearity of radial basis neural networks. Predictive regions which used the width of confidence intervals as the nonconformity measure gave much better results. But those results were not as consistent as the results of the other methods. Predictive regions based on the local modeling of prediction errors gave us good results and the computation of the regions was very fast. A smaller number of neighbors gave better results for these regions. The best results were achieved by the regions based on the variance of a bagged model. The only drawback of this method is that a lot of models must be fitted and it is, therefore, computationally very inefficient.

References

1. Z. Bosnic, Kononenko, I.: *Comparison of approaches for estimating reliability of individual regression predictions.* Data & Knowledge Engineering, 2008, 504–516.
2. A. Gammerman, G. Shafer, V. Vovk: *Algorithmic learning in a random world.* Springer Science+Business Media, 2005.
3. E. Uusipaikka: *Confidence intervals in generalized regression models.* Chapman & Hall, 2009.
4. H. Papadopoulos, V. Vovk, A. Gammerman: *Regression conformal prediction with nearest neighbours.* Journal of Artificial Intelligence Research 40, 2011, 815–840.
5. S. Valero, E. Argente, et al.: *DoE framework for catalyst development based on soft computing techniques.* Computers and Chemical Engineering 33(1), 2009, 225–238.

Fuzzy classification rules based on similarity[*]

Martin Holeňa[1] and David Štefka[2]

[1] Institute of Computer Science, Academy of Sciences of the Czech Republic
Pod Vodárenskou věží 2, 182 07 Prague
[2] Faculty of Nuclear Science and Physical Engineering
Czech Technical University
Trojanova 13, 120 00 Prague

Abstract. *The paper deals with the aggregation of classification rules by means of fuzzy integrals, in particular with the fuzzy measures employed in that aggregation. It points out that the kinds of fuzzy measures commonly encountered in this context do not take into account the diversity of classification rules. As a remedy, a new kind of fuzzy measures is proposed, called similarity-aware measures, and several useful properties of such measures are proven. Finally, results of extensive experiments on a number of benchmark datasets are reported, in which a particular similarity-aware measure was applied to a combination of Choquet or Sugeno integrals with three different ways of creating ensembles of classification rules. In the experiments, the new measure was compared with the traditional Sugeno λ-measure, to which it was clearly superior.*

1 Introduction

Logical formulas of specific kinds, usually called *rules*, are a traditional way of formally representing knowledge. Therefore, it is not surprising that they are also the most frequent representation of the knowledge discovered in data mining.

The most natural base for differentiating between existing rules extraction methods is the *syntax and semantics of the extracted rules* [10]. Syntactical differences between them are, however, not very deep because, principally, any rule r from a ruleset $\mathcal{R}$ has one of the forms $S_r \sim S'_r$, or $A_r \to C_r$, where S_r, S'_r, A_r and C_r are formulas of the considered logic, and $\sim$, $\to$ are symbols of the language of that logic. The difference between both forms concerns semantic properties of the symbols $\sim$ and $\to$: $S_r \sim S'_r$ is symmetric with respect to S_r, S'_r in the sense that its validity always coincides with that of $S'_r \sim S_r$ whereas $A_r \to C_r$ is not symmetric with respect to A_r, C_r in that sense. In the case of a propositional logic, $\sim$ and $\to$ are the connectives equivalence ($\equiv$) and implication, respectively, whereas in the case of a predicate logic, they are generalized quantifiers. To distinguish the formulas involved in the asymmetric case, A_r is called *antecedent* and C_r *consequent* of r.

More important is the semantic of the rules (cf. [5]), especially the difference between *rules of the Boolean logic* and *rules of a fuzzy logic*. Due to the semantics of Boolean and fuzzy formulas, the former are valid for crisp sets of objects, whereas the validity of the latter is a fuzzy set on the universe of all considered objects. Boolean rulesets are extracted more frequently, especially some specific types of them, such as *classification rulesets* [6, 9]. Those are sets of implications such that $\{A_r\}_{r\in\mathcal{R}}$ and $\{C_r\}_{r\in\mathcal{R}}$ partition the set $\mathcal{O}$ of considered objects, where $\{\cdot\}_{r\in\mathcal{R}}$ stands for the set of distinct formulas in $(\cdot)_{r\in\mathcal{R}}$. Abandoning the requirement that $\{A_r\}_{r\in\mathcal{R}}$ partitions $\mathcal{O}$ (at least in the sense of a crisp partitioning) allows to generalize those rulesets also to fuzzy antecedents [15]. For Boolean antecedents, however, this requirement entails a natural definition of the validity of a whole classification ruleset $\mathcal{R}$ for an object x. Assuming that all information about x conveyed by $\mathcal{R}$ is conveyed by the single rule r covering x (i.e., with A_r valid for x), the validity of $\mathcal{R}$ for x can be defined to coincide with the validity of $A_r \to C_r$ for that r, which in turn equals the validity of C_r for x.

It is also possible to combine several existing classification rules into a new one. Such aggregation can be either *static*, i.e., the result is the same for all inputs, or *dynamic*, where it is adapted to the currently classified input [11, 19]. In the aggregation of classification rules, we usually try to create a team of rules that are not similar. This property is called *diversity* [14]. There are many methods for building a diverse team of classifiers [2, 3, 16].

One of popular aggregation operators is the *fuzzy integral* [7, 12, 13, 17]. It aggregates the outputs of the individual classification rules with respect to a fuzzy measure. The role of fuzzy measures in the aggregation of classification rules, in particular their role with respect to the diversity of the rules, was the subject of the research reported in this paper.

The following section recalls the fuzzy integrals and fuzzy measures encountered in the aggregation of classification rules. In Section 3, which is the key section of the paper, a new fuzzy measure, called similarity-aware measure, is introduced and its theoretical properties are studied. Finally, in Section 4, results of ex-

[*] The research reported in this paper has been supported by the Czech Science Foundation (GA ČR) grant P202/11/1368.

tensive experiments and comparison with the traditional Sugeno λ-measure are reported.

2 Fuzzy integrals and measures in classification rules aggregation

Several definitions of a fuzzy integral exists in the literature – among them, the Choquet integral and the Sugeno integral are used most often. The role played in usual integration by additive measures (such as probability or Lebesgue measure) is in fuzzy integration played by fuzzy measures. In this section, basic concepts pertaining to different kinds of fuzzy measures will be recalled, as well as the definitions of Choquet and Sugeno integrals. Due to the intended context of aggregation of classification rules, we restrict attention to $[0, 1]$-valued functions on finite sets.

Definition 1. *A* fuzzy measure *μ on a finite set $\mathcal{U} = \{u_1, \ldots, u_r\}$ is a function on the power set of $\mathcal{U}$,*

$$\mu : \mathcal{P}(\mathcal{U}) \to [0, 1] \tag{1}$$

fulfilling:

1. the boundary conditions

$$\mu(\emptyset) = 0, \mu(\mathcal{U}) = 1 \tag{2}$$

2. the monotonicity

$$A \subseteq B \Rightarrow \mu(A) \leq \mu(B) \tag{3}$$

The values $\mu(u_1), \ldots, \mu(u_r)$ are called fuzzy densities.

Definition 2. *The* Choquet integral *of a function $f : \mathcal{U} \to [0, 1]$, $f(u_i) = f_i$, $i = 1, \ldots, r$, with respect to a fuzzy measure μ is defined as:*

$$(\mathrm{Ch}) \int f d\mu = \sum_{i=1}^{r} (f_{<i>} - f_{<i-1>}) \mu(A_{<i>}), \tag{4}$$

where $< \cdot >$ indicates that the indices have been permuted, such that $0 = f_{<0>} \leq f_{<1>} \leq \cdots \leq f_{<r>} \leq 1$. $A_{<i>} = \{u_{<i>}, \ldots, u_{<r>}\}$ denotes the set of of elements of $\mathcal{U}$ corresponding to the $(r - i + 1)$ highest values of f.

Definition 3. *The* Sugeno integral *of a function $f : \mathcal{U} \to [0, 1]$, $f(u_i) = f_i$, $i = 1, \ldots, r$, with respect to a fuzzy measure μ is defined as:*

$$(\mathrm{Su}) \int f d\mu = \max_{i=1}^{r} \min(f_{<i>}, \mu(A_{<i>})). \tag{5}$$

To define a general fuzzy measure in the discrete case, we need to define all its 2^r values, which is usually very complicated. To overcome this weakness, measures which do not need all the 2^r values have been developed [7, 17]:

Definition 4. *A fuzzy measure μ on $\mathcal{U}$ is called* symmetric *if*

$$|A| = |B| \Rightarrow \mu(A) = \mu(B) \tag{6}$$

$$\textit{for } A, B \subseteq \mathcal{U}, \tag{7}$$

where $| \cdot |$ denotes the cardinality of a set.

Consequently, the value of a symmetric measure depends only on the cardinality of its argument. If a symmetric measure is used in Choquet integral, the integral reduces to the ordered weighted average operator [17]. However, symmetric measures assume that all elements of $\mathcal{U}$ have the same importance, thus they do not take into account the diversity of elements.

Definition 5. *Let $\perp$ be a t-conorm. A fuzzy measure μ is called* $\perp$-decomposable *if*

$$\mu(A \cup B) = \mu(A) \perp \mu(B) \quad \textit{for disjoint } A, B \subseteq \mathcal{U} \tag{8}$$

Hence, $\perp$-decomposable measures need only the r fuzzy densities, whereas all the other values are computed using the formula (8). Particular cases of this kind of fuzzy measures are *additive measures*, including probabilistic measures ($\perp$ being the bounded sum), and the Sugeno λ-measure.

Definition 6. Sugeno λ-measure *[7, 17] on a finite set $\mathcal{U} = \{u_1, \ldots, u_r\}$ is defined*

$$\mu(A \cup B) = \mu(A) + \mu(B) + \lambda\mu(A)\mu(B), \tag{9}$$

for disjoint $A, B \in \mathcal{U}$, and some fixed $\lambda > -1$. The value of λ is:

a) computed as the unique non-zero root greater than -1 of the equation

$$\lambda + 1 = \prod_{i=1,\ldots,r} (1 + \lambda\mu(\{u_i\})) \tag{10}$$

if the densities do not sum up to 1;
b) $\lambda = 0$ else.

If the densities sum up to 1, the fuzzy measure is additive. Sugeno λ measure is a $\perp$-decomposable measure for the t-norm

$$x \perp y = \min(1, x + y + \lambda xy). \tag{11}$$

A serious weakness of any $\perp$-decomposable measure is that the fuzzy measure of a set of two (or more) classification rules is fully determined by the formula (8) for a fixed $\perp$. Therefore, if interactions between elements are to be taken into account, then they have to be incorporated directly into the fuzzy measure. That fact motivated our attempt to elaborate the concept of similarity-aware fuzzy measures.

3 Similarity-aware measures and their properties

Before introducing similarity-aware measures, let us first recall the notion of similarity [8].

Definition 7. *Let $\wedge$ be a t-norm and let $\sim : \mathcal{U} \times \mathcal{U} \to [0,1]$ be a fuzzy relation. $\sim$ is called a* similarity on $\mathcal{U}$ with respect to $\wedge$ *if the following holds for $a, b, c \in \mathcal{U}$:*

$$\sim(a,a) = 1 \text{ (reflexivity)}, \tag{12}$$

$$\sim(a,b) = \sim(b,a) \text{ (symmetry)}, \tag{13}$$

$$\sim(a,b) \wedge \sim(b,c) \leq \sim(a,c) \text{ (transitivity w.r.t. } \wedge \text{)}. \tag{14}$$

In the context of aggregation of crisp classification rules, we will work with an empirically defined relation, which, for rules ϕ_k, ϕ_l, is defined as the proportion of equal consequents on some validation set of patterns $\mathcal{V} \subset \mathcal{O}$,

$$\sim(\phi_k, \phi_l) = \frac{\sum_{x \in \mathcal{V}} I(C_{\phi_k}(x) = C_{\phi_l}(x))}{|\mathcal{V}|}. \tag{15}$$

It is easily seen that the relation (15) is a similarity with respect to the Łukasiewicz t-norm

$$\wedge_L(a,b) = \max(a+b-1, 0), \tag{16}$$

but it is not a similarity with respect to the standard (minimum, Gödel) t-norm

$$\wedge_S(a,b) = \min(a,b), \tag{17}$$

or the product t-norm

$$\wedge_P(a,b) = ab. \tag{18}$$

Fuzzy integral represents a convenient tool to work with the diversity of classification rules: As we are computing the fuzzy measure values $\mu(A_{<i>})$, we are considering a single rule $\phi_{<i>}$ at each step i, and therefore we can influence the increase of the fuzzy measure based on the similarity of $\phi_{<i>}$ to the set of rules already involved in the integration, i.e., $A_{<i+1>} = \{\phi_{<i+1>}, \dots, \phi_{<r>}\}$. If $\phi_{<i>}$ is similar to the classifiers in $A_{<i+1>}$, the increase in the fuzzy measure should be small (since the importance of the set $A_{<i>}$ should be similar to the importance of the set $A_{<i+1>}$), and if $\phi_{<i>}$ is not similar to the classifiers in $A_{<i+1>}$, the increase of the fuzzy measure should be large. These ideas motivated the following definition:

Definition 8. *Let $\mathcal{U} = \{u_1, \dots, u_r\}$ be a set, let $\sim$ be a similarity w.r.t. a t-norm $\wedge$, and let S be a an $r \times r$ matrix such that:*

$$S = (s_{i,j})_{i,j=1}^{r} \text{ with } s_{i,j} = \sim(u_i, u_j). \tag{19}$$

Let further $\kappa_i \in [0,1]$, $i = 1, \dots, r$ denote some kind of weight (confidence, importance) of u_i, and let $[\cdot]$ denote index ordering according to κ, such that $0 \leq \kappa_{[1]} \leq \dots \leq \kappa_{[r]} \leq 1$. Finally, let

$$\tilde{\mu}^{(S)} : \mathcal{P}(\mathcal{U}) \to [0, \infty) \tag{20}$$

be a mapping such that for $X \subseteq \mathcal{U}$,

$$\tilde{\mu}^{(S)}(X) = \sum_{i=1}^{r} I(u_{[i]} \in X) \kappa_{[i]} (1 - \max_{j=i+1}^{r} s_{[i],[j]}), \tag{21}$$

where we define $\max_{j=r+1}^{r} s_{[r],[j]} = 0$, and I denotes the indicator of thruth value, i.e.,

$$I(true) = 1, I(false) = 0. \tag{22}$$

Then the mapping

$$\mu^{(S)} : \mathcal{P}(\mathcal{U}) \to [0,1], \text{ defined} \tag{23}$$

$$\mu^{(S)}(X) = \frac{\tilde{\mu}^{(S)}(X)}{\tilde{\mu}^{(S)}(\mathcal{U})}, \tag{24}$$

is called a similarity-aware measure based on S.

Proposition 1. *$\mu^{(S)}$ is a fuzzy measure on $\mathcal{U}$.*

Proof. The boundary conditions follow directly from the definition of $\mu^{(S)}$. For the monotonicity, let $A \subseteq B$; then

$$\tilde{\mu}^{(S)}(A) = \sum_{i=1}^{r} I(u_{[i]} \in A) \kappa_{[i]} (1 - \max_{j=i+1}^{r} s_{[i],[j]}) \leq$$
$$\leq \sum_{i=1}^{r} I(u_{[i]} \in B) \kappa_{[i]} (1 - \max_{j=i+1}^{r} s_{[i],[j]}) =$$
$$= \tilde{\mu}^{(S)}(B), \tag{25}$$

due to $I(u_{[i]} \in A) = 1 \Rightarrow I(u_{[i]} \in B) = 1$.

Proposition 2. *For any of the 2^r subsets $X \subset \mathcal{U}$, the value $\mu(X)$ can be expressed simply as the sum of values of μ on singletons*

$$\mu^{(S)}(X) = \sum_{u_i \in X} \mu^{(S)}(u_i). \tag{26}$$

Proof. According to (21) and (23), the value of μ on the singletosn $u_i, i = 1, \dots, r$ is

$$\mu^{(S)}(u_i) = \frac{1}{\tilde{\mu}^{(S)}(\mathcal{U})} \kappa_{[i]} (1 - \max_{j=i+1}^{r} s_{[i],[j]}). \tag{27}$$

Then (26) follows directly from (21).

The following propositions show that if for some i, the i-th classification rule is totally similar to some other rule in $A_{<i+1>}$, then $\mu^{(S)}$ does not increase, and if it is totally unsimilar to all classifiers in $A_{<i+1>}$, the increase in $\mu^{(S)}$ is maximal.

Proposition 3. *Let $f : \mathcal{U} \to [0,1]$, and let the matrix S in (19) fulfills*

$$s_{i,j} = 1 \textit{ for } i \neq j. \tag{28}$$

Then:

1. $(\forall X \subseteq \mathcal{U})\ u_{[r]} \in X \Rightarrow \mu^{(S)} = 1$,
2. $(\forall X \subseteq \mathcal{U})\ u_{[r]} \notin X \Rightarrow \mu^{(S)} = 0$,
3. $(\mathrm{Ch}) \int f d\mu^{(S)} = (\mathrm{Su}) \int f d\mu^{(S)} = f_{[r]}$.

Proof. 1. and 2. follow directly from the fact that

$$\max_{j=i+1}^{r} s_{[i],[j]} = \begin{cases} 0 \text{ for } i = r, \\ 1 \text{ for } i < r. \end{cases} \tag{29}$$

and therefore

$$\tilde{\mu}^{(S)} = I(u_{[r]} \in X)\kappa_{[r]}. \tag{30}$$

We will prove 3. only for the Choquet integral, the case of Sugeno integral is analogous. Let $j \in \{1, \dots, r\}$ such that $< j >= [r]$; then $(\forall i > j)\ u_{[r]} \notin A_{<i>}$, and therefore $\mu^{(S)}(A_{<i>}) = 0$; $(\forall i \leq j)\ u_{[r]} \in A_{<i>}$, and therefore $\mu^{(S)} = 1$. Using this in the definition of the Choquet integral, we obtain

$$\begin{aligned}(\mathrm{Ch}) \int f d\mu^{(S)} = \\ = \sum_{i=1}^{r} (f_{<i>} - f_{<i-1>})\mu^{(S)}(A_{<i>}) = \\ = \sum_{i=1}^{j} (f_{<i>} - f_{<i-1>}) = \\ = f_{<j>} = f_{[r]}.\end{aligned} \tag{31}$$

Proposition 4. *Let $f : \mathcal{U} \to [0,1]$, and let the matrix S in (19) fulfills $s_{i,j} = 0$ for $i \neq j$. Then:*

1. $(\forall X \subseteq \mathcal{U})\ \mu^{(S)} = \frac{\sum_{i:u_{[i]} \in X} \kappa_{[i]}}{\sum_{i=1}^{r} \kappa_i}$,
2. $(\mathrm{Ch}) \int f d\mu^{(S)} \mu^{(S)} = \frac{\sum_{i=1}^{r} \kappa_i f_i}{\sum_{i=1}^{r} \kappa_i}$,
3. $(\mathrm{Su}) \int f d\mu^{(S)} = \max_{k=1}^{r}(f_{<k>}, \frac{\sum_{i=k}^{r} \kappa_{<i>}}{\sum_{i=1}^{r} \kappa_i})$.

Proof. 1. follows directly from the definition of similarity-aware measure, and 2. and 3. are applications of 1. to the definition of the Choquet/Sugeno integral.

4 Experimental testing

We have experimentally compared the performance of the proposed measure with the Sugeno λ-measure for the aggregation of classification rules by fuzzy integrals (Choquet, Sugeno). The ensembles have been created as random forests from rules obtained with classification trees [3], by bagging [2] from rules obtained with k-NN classifiers, and by the multiple feature subset method [1] from rules obtained with quadratic discriminant analysis.

In this section, we present results of comparing the measures using 10-fold crossvalidation on 5 artificial and 11 real-world datasets (the properties of the datasets are shown in Table 1). For the random forests, the number of trees was set to $r = 20$, the number of features to explore in each node varied between 2 and 5 (depending on the dimensionality of the particular dataset), the maximal size of a leaf was set to 10 (see [3] for description of the parameters). For the QDA and k-NN based ensembles, their size was set also to $r = 20$, and we used $k = 5$ as the number of neighbors for k-NN classifiers. As the weights $\kappa_1, \dots, \kappa_r$ of the classification rules, we used

$$\kappa_i(\phi) = \frac{\sum\limits_{x \in \mathcal{V}(A_\phi)} I(C'_\phi(x) = C_\phi(x))}{|\mathcal{V}(A_\phi)|}, \tag{32}$$

where $\mathcal{V}(A_\phi) \subseteq \mathcal{V}$ is the set of validation patterns belonging to some kind of neighborhood of A_ϕ. For example, if A_ϕ concerns values of vectors in an Euclidean space, then $\mathcal{V}(A_\phi)$ is the set of k nearest neighbors under Euclidean metric of the set where the antecedent A_ϕ is valid. The number of neighbors was set to 5, 10, or 20, depending on the size of the dataset.

Table 2 shows the results of the performed comparisons. We also measured the statistical significance of the pairwse improvements (using the analysis of variance on the 5% confidence level by the Tukey-Kramer method).

We interpret the results presented in Table 2 as a confirmation of the usefulness of similarity-aware fuzzy measures proposed in Definition 8.

5 Conclusion

In this paper, we have studied the application of the fuzzy integral as an aggregation operator for classification rules in the context of their similarities. We have shown that traditionally used symmetric, or additive and other $\perp$-decomposable measures are not a good choice for combining classification rules by fuzzy integral and we have defined similarity-aware measures, which take into account both the confidence / importance and the similarities of the aggregated rules. We have shown some basic theoretical properties and special cases of the measures, including the fact that apart the singletons, the 2^r values of μ are obtained using only summation. In addition, we have experimentally compared the performance of the measures to the Sugeno λ-measure using Choquet and Sugeno fuzzy integrals on 16 benchmark datasets for 3 different ways

dataset	nr. of patterns	nr. of classes	dimension
Artificial			
clouds [4]	5000	2	2
concentric [4]	2500	2	2
gauss 3D [4]	5000	2	3
ringnorm [18]	3000	2	20
waveform [18]	5000	3	21
Real-world			
glass [18]	214	7	9
letters [18]	20000	26	16
pendigits [18]	10992	10	16
phoneme [4]	5427	2	5
pima [18]	768	2	8
poker [18]	4828	3	10
satimage [4]	6435	6	4
transfusion [18]	748	2	4
vowel [18]	990	11	10
wine [18]	178	3	13
yeast [18]	1484	4	8

Table 1. Datasets used in the experiments.

of obtaining ensembles of classification rules. The experimental comparison clearly supports our theoretical conjecture that similarity-aware measures are more suitable for the aggregation of classification rules than traditionally used additive and $\perp$-decomposable fuzzy measures.

References

1. S. D. Bay: *Nearest neighbor classification from multiple featre subsets.* Intelligent Data Analysis 3, 1999, 191–209.
2. L. Breiman: *Bagging predictors.* Machine Learning 24, 1996, 123–140.
3. L. Breiman: *Random forests.* Machine Learning 45, 2001, 5–32.
4. Machine Learning Group Catholic University of Leuven. Elena database. `http://mlg.info.ucl.ac.be/index.php?page=Elena`.
5. D. Dubois, Hüllermeier, H. Prade: *A systematic approach to the assessment of fuzzy association rules.* Data Mining and Knowledge Discovery 13, 2006, 167–192.
6. L. Geng, H. J. Hamilton: *Choosing the right lens: Finding what is interesting in data mining.* In F. Guillet and H. J. Hamilton, (Eds), Quality Measures in Data Mining, Springer Verlag, Berlin, 2007, 3–24.
7. M. Grabisch, H. T. Nguyen, E. A. Walker: *Fundamentals of uncertainty calculi with applications to fuzzy inference.* Kluwer Academic Publishers, Dordrecht, 1994.
8. P. Hájek: *Metamathematics of fuzzy logic.* Kluwer Academic Publishers, Dordrecht, 1998.
9. D. J. Hand: *Construction and assessment of classification rules.* John Wiley and Sons, New York, 1997.
10. M. Holeňa: *Measures of ruleset quality capable to represent uncertain validity.* Submitted to International Journal of Approximate Reasoning.
11. A. H. R. Ko, R. Sabourin, A. S. Britto: *From dynamic classifier selection to dynamic ensemble selection.* Pattern Recognition 41, 2008, 1718–1731.
12. L. I. Kuncheva: *Fuzzy versus nonfuzzy in combining classifiers designed by boosting.* IEEE Transactions on Fuzzy Systems 11, 2003, 729–741.
13. L. I. Kunchev: *Combining pattern classifiers: methods and algorithms.* John Wiley and Sons, New York, 2004.
14. L. I. Kuncheva C. J. Whitaker: *Measures of diversity in classifier ensembles.* Machine Learning 51, 2003, 181–207.
15. L. E. Peterson, M. A. Coleman: *Machine learning based receiver operating characteristic (ROC) curves for crisp and fuzzy classification of DNA microarrays in cancer research.* International Journal of Approximate Reasoning 47, 2008, 17–36.
16. L. Rokach: *Taxonomy for characterizing ensemble methods in classification tasks: A review and annotated bibliography.* Computational Statistics and Data Analysis 53, 2009, 4046–4072.
17. V. Torra, Y. Narukawa: *Modeling decisions: information fusion and aggregation operators.* Springer Verlag, Berlin, 2007.
18. Machine Learning Group University of California Irwine. Repository of machine learning databases. `http://www.ics.uci.edu/ mlearn/MLRepository.html`.
19. D. Štefka, M. Holeňa: *Dynamic classifier systems and their applications to random forest ensembles.* In Adaptive and Natural Computing Algorithms. Lecture Notes in Computer Science 5495, Springer Verlag, Berlin, 2009, 458–468.

dataset	Choquet integral		Sugeno integral	
	λ-measure	μ^S	λ-measure	μ^S
random forests				
clouds	12.40 ± 1.81	**12.25 ± 1.85**	12.80 ± 1.64	12.33 ± 1.47
concentric	4.32 ± 1.25	**2.82 ± 1.30**	3.24 ± 1.37	2.98 ± 1.50
gauss-3D	23.92 ± 2.97	**22.76 ± 1.59**	24.60 ± 1.36	23.28 ± 1.58
glass	21.3 ± 10.3	**14.1 ± 3.5**	24.1 ± 7.0	17.5 ± 9.4
letters	**7.1 ± 0.6**	7.3 ± 0.2	8.0 ± 0.6	7.9 ± 0.8
pendigits	3.1 ± 0.5	**2.7 ± 0.5**	3.2 ± 0.4	3.8 ± 0.7
phoneme	**12.4 ± 1.2**	13.2 ± 1.9	12.7 ± 0.8	13.3 ± 1.6
pima	26.0 ± 4.8	**23.8 ± 2.0**	25.0 ± 2.2	23.9 ± 3.6
poker	46.5 ± 3.0	**44.4 ± 1.3**	46.5 ± 1.5	45.1 ± 1.9
ringnorm	13.27 ± 2.11	*7.69 ± 2.06*	12.74 ± 2.08	***7.46 ± 1.79***
satimage	14.7 ± 1.4	**14.3 ± 1.3**	14.9 ± 0.9	14.8 ± 1.4
transfusion	4.8 ± 1.1	**2.3 ± 0.7**	4.9 ± 1.0	2.6 ± 0.7
vowel	14.5 ± 3.0	**13.1 ± 3.5**	17.0 ± 5.3	13.4 ± 3.8
waveform	18.56 ± 2.42	**17.93 ± 1.89**	18.24 ± 3.04	18.23 ± 1.58
wine	5.6 ± 6.0	**3.3 ± 5.5**	3.4 ± 4.0	6.6 ± 5.8
yeast	38.2 ± 4.1	**34.8 ± 2.6**	38.5 ± 3.7	36.3 ± 3.4
k-NN classifiers				
clouds	**11.93 ± 2.29**	12.12 ± 1.57	12.64 ± 2.48	12.96 ± 2.26
concentric	1.39 ± 0.77	1.72 ± 0.57	**1.30 ± 0.80**	1.56 ± 0.64
gauss-3D	26.71 ± 2.55	**26.00 ± 2.88**	27.68 ± 3.66	26.28 ± 2.74
glass	22.4 ± 9.8	20.7 ± 10.3	21.7 ± 11.1	**19.3 ± 6.5**
letters	17.7 ± 2.7	**17.6 ± 2.9**	19.3 ± 3.1	19.1 ± 2.7
pendigits	1.3 ± 0.8	**1.4 ± 0.8**	1.3 ± 0.5	1.3 ± 0.7
phoneme	14.6 ± 0.9	**14.2 ± 2.4**	14.4 ± 1.8	14.5 ± 1.7
pima	**29.1 ± 5.1**	30.2 ± 7.2	29.5 ± 4.4	30.3 ± 6.6
poker	45.3 ± 2.4	**43.5 ± 2.3**	47.2 ± 2.7	43.9 ± 1.4
ringnorm	36.20 ± 4.41	**34.28 ± 2.59**	33.56 ± 3.34	33.48 ± 2.94
satimage	16.5 ± 2.0	**15.5 ± 1.7**	16.8 ± 2.4	16.2 ± 2.3
transfusion	24.0 ± 4.0	**23.4 ± 4.7**	25.2 ± 3.5	24.0 ± 4.7
vowel	4.8 ± 2.2	**4.0 ± 1.9**	5.6 ± 2.1	7.0 ± 1.8
waveform	19.40 ± 2.10	**18.28 ± 2.85**	19.57 ± 2.20	19.04 ± 2.99
wine	30.0 ± 10.3	**28.6 ± 14.8**	31.2 ± 6.7	33.4 ± 16.0
yeast	41.8 ± 4.7	**40.5 ± 2.6**	42.6 ± 3.7	40.7 ± 3.6

QDA with multiple subsets				
clouds	26.00 ± 2.70	23.14 ± 2.49	25.74 ± 1.92	***22.66 ± 1.10***
concentric	4.36 ± 1.96	3.68 ± 1.68	5.72 ± 1.84	**3.40 ± 0.98**
gauss-3D	23.87 ± 1.86	**22.06 ± 2.10**	23.96 ± 2.03	22.36 ± 2.12
glass	42.3 ± 10.9	38.5 ± 12.0	43.2 ± 14.9	**32.4 ± 12.5**
letters	17.1 ± 0.7	***14.7 ± 0.7***	17.2 ± 0.7	*14.7 ± 0.8*
pendigits	2.8 ± 0.5	**2.2 ± 0.2**	2.7 ± 0.5	2.7 ± 0.6
phoneme	25.4 ± 2.4	*20.8 ± 1.4*	24.7 ± 1.0	***20.2 ± 2.2***
pima	27.9 ± 4.7	**25.5 ± 4.2**	28.3 ± 3.3	26.1 ± 5.1
poker	66.1 ± 2.1	*55.1 ± 2.3*	66.3 ± 3.9	***55.1 ± 2.1***
ringnorm	1.94 ± 0.96	2.53 ± 1.01	**1.68 ± 0.60**	3.66 ± 1.31
satimage	17.0 ± 1.3	**15.7 ± 1.1**	17.2 ± 2.0	16.2 ± 1.3
transfusion	29.6 ± 8.6	**22.3 ± 4.5**	29.2 ± 7.1	23.4 ± 3.4
vowel	16.7 ± 5.4	**14.0 ± 3.8**	18.1 ± 4.1	15.6 ± 3.3
waveform	15.73 ± 2.07	**14.52 ± 1.59**	15.33 ± 1.72	14.54 ± 1.80
wine	1.2 ± 2.5	2.8 ± 3.9	**0.6 ± 1.9**	3.3 ± 2.9
yeast	49.0 ± 4.3	*39.8 ± 4.3*	49.5 ± 4.9	***39.1 ± 3.8***

Table 2. Mean error rates ± standard deviation of the error rate [%], based on 10-fold crossvalidation. The best result for each dataset is displayed in boldface, statistically significant improvements (measured by the analysis of variance using the Tukey-Kramer method at the 5% level) are displayed in italics

Evolutionary optimization with active learning of surrogate models and fixed evaluation batch size*

Viktor Charypar[1] and Martin Holeňa[2]

[1] Czech Technical University
Faculty of Nuclear Sciences and Physical Engineering
Břehová 7, 115 19 Praha 1, Czech Republic
charyvik@fjfi.cvut.cz
[2] Institute of Computer Science
Academy of Sciences of the Czech Republic
Pod vodárenskou věží 2, 182 07 Praha, Czech Republic
martin@cs.cas.cz

Abstract. *Evolutionary optimization is often applied to problems, where simulations or experiments used as the fitness function are expensive to run. In such cases, surrogate models are used to reduce the number of fitness evaluations. Some of the problems also require a fixed size batch of solutions to be evaluated at a time. Traditional methods of selecting individuals for true evaluation to improve the surrogate model either require individual points to be evaluated, or couple the batch size with the EA generation size. We propose a queue based method for individual selection based on active learning of a kriging model. Individuals are selected using the confidence intervals predicted by the model, added to a queue and evaluated once the queue length reaches the batch size. The method was tested on several standard benchmark problems. Results show that the proposed algorithm is able to achieve a solution using significantly less evaluations of the true fitness function. The effect of the batch size as well as other parameters is discussed.*

1 Introduction

Evolutionary optimization algorithms are a popular class of optimization techniques suitable for various optimization problems. One of their main advantages is the ability to find optima of black-box functions – functions that are not explicitly defined and only their input/output behavior is known from previous evaluations of a finite number of points in the input space. This is typical for applications in engineering, chemistry or biology, where the evaluation is performed in a form of computer simulation or physical experiment.

The main disadvantage for such applications is the very high number of evaluations of the objective function (called fitness function in the evolutionary optimization context) needed for an evolutionary algorithm (EA) to reach the optimum. Even if the simulation used as the objective function takes minutes to finish, the traditional approach becomes impractical. When the objective function is evaluated using a physical experiment, in the evolutionary optimization of catalytic materials [1] for example, an evaluation for one generation of the algorithm takes between several days and several weeks and costs thousands of euros.

The typical solution to this problem is performing only a part of all evaluations using the true fitness function and using a response-surface model as its replacement for the rest. This approach is called surrogate modeling. When using a surrogate model, only a small portion of all the points that need to be evaluated is evaluated using the true objective function (simulation or experiment) and for the rest, the model prediction is assigned as the fitness value. The model is built using the information from the true fitness evaluations.

Since the fitness function is assumed to be highly non-linear the modeling methods used are non-linear as well. Some of the commonly used methods include artificial neural networks, radial basis functions, regression trees, support vector machines or Gaussian processes [3].

Furthermore, some experiments require a fixed number of samples to be processed at one time. This presents its own set of challenges for adaptive sampling and is the main concern of this paper. We present an evolutionary optimization method assisted by a variant of a Gaussian-process-based interpolating model called kriging. In order to best use the evaluation budget, our approach uses active learning methods in selecting individuals to evaluate using the true fitness function. A key feature of the approach is support for online and offline batch evaluation with arbitrary batch size independent of the generation size of the EA.

The rest of the paper is organized as follows: in the following section we introduce the kriging surrogate model and its properties, in section 2 the methods of

* This work was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS12/196/OHK3/3T/14 as well as the Czech Science Foundation grant 201/08/0802.

coupling a model to the evolutionary optimization are discussed, section 4 provides details of the proposed method and finally, the results of testing the method are presented and discussed in section 5.

2 Model-assisted evolutionary optimization

Since the surrogate model used as a replacement for the fitness function in the EA is built using the results of the true fitness function evaluations, there are two competing objectives. First, we need to get the most information about the underlying relations in the data, in order to build a precise model of the fitness function. If the model does not capture the features of the fitness function correctly, the optimization can get stuck in a fake optimum or generally fail to converge to a global one. Second, we have a limited budget for the true fitness function evaluations. Using many points from the input space to build a perfect model can require more true fitness evaluations than not employing a model at all.

In the general use of surrogate modeling, such as design space exploration, the process of selecting points from the input space to evaluate and build the model upon is called sampling [3]. Traditionally, the points to sample are selected upfront. Upfront sampling schemes are based on the theory of design of experiments (DoE), e.g. a Latin hypercube design. When we don't know anything about the function we are trying to model, it is better to use a small set of points as a base for an initial model, which is then iteratively improved using new samples, selected based on the information from previous function evaluations and the model itself. This approach is called adaptive sampling [3].

Using the surrogate model in an evolutionary optimization algorithm, the adaptive sampling decisions change from selecting which points of the input space to evaluate in order to improve the model to whether to evaluate a given point (selected by the EA) with the true fitness function or not. There are two general approaches to this choice: the generation-based approach and the individual-based approach. We will discuss both, with emphasis on the latter, a variant of which is used in the method we propose in section 4.

2.1 Generation-based approach

In the generation-based approach the decision whether to evaluate an individual point with the true fitness function is made for the whole generation of the evolutionary algorithm. The optimization takes the following steps.

1. An initial N_i generations of the EA is performed, yielding sets $\mathcal{G}_1, \ldots, \mathcal{G}_{N_i}$ of individuals $(\mathbf{x}, f_t(\mathbf{x}))$, f_t being the true fitness function.
2. The model M is trained on the individuals $(\mathbf{x}, f_t(\mathbf{x})) \in \bigcup_{i=1}^{N_i} \mathcal{G}_i$.
3. The fitness function f_t is replaced by a model prediction f_M.
4. T generations are performed evaluating f_M as the fitness function.
5. One generation is performed using f_t yielding a set $\mathcal{G}_j$ of individuals. (initially $j = N_i + 1$)
6. The model is retrained on the individuals $(\mathbf{x}, f_t(\mathbf{x})) \in \bigcup_{i=1}^{j} \mathcal{G}_i$
7. Steps 4–6 are repeated until the optimum is reached.

The amount of true fitness evaluations in this approach is dependent on the population size of the EA and the frequency of control generations T, which can be fixed or adaptively changed during the course of the optimization [6]. For problems requiring batched evaluation this approach has the advantage of evaluating the whole generation, the size of which can be set to the size of the evaluation batch. The main disadvantage of the generation-based strategy is that not all individuals in the control generation are necessarily beneficial to the model quality and the expensive true fitness evaluations are wasted.

2.2 Individual-based approach

As opposed to the generation-based approach, in the individual-based strategy, the decision whether to evaluate a given point using the true fitness function or the surrogate model is made for each individual separately.

In model-based optimization in general, there are several possible approaches to individual-based sampling. The most used approach in the evolutionary optimization is pre-selection. In each generation of the EA, number of points, which is a multiple of the population size, is generated and evaluated using the model prediction. The best of these individuals form the next generation of the algorithm. The optimization is performed as follows.

1. An initial set of points $\mathcal{S}$ is chosen and evaluated using the true fitness function f_t.
2. Model M is trained using the pairs $(\mathbf{x}, f_t(\mathbf{x})) \in \mathcal{S}$
3. A generation of the EA is run with the fitness function replaced by the model prediction f_M and a population $\mathcal{O}_i$ of size qp is generated and evaluated with f_M, where p is the desired population size for the EA and q is the pre-screening ratio. Initially, $i = 1$.

4. A subset $\mathcal{P} \subset \mathcal{O}$ is selected according to a selection criterion.
5. Individuals from $\mathcal{P}$ are evaluated using the true fitness function f_t.
6. The model M is retrained using $\mathcal{S} \cup \mathcal{P}$, the set $\mathcal{S}$ is replaced with $\mathcal{S} \cup \mathcal{P}$, and the EA resumes from step 3.

Another possibility, called the best strategy [5], is to replace $\mathcal{S}$ with $\mathcal{S} \cup \mathcal{O}$ instead of just $\mathcal{P}$ in step 6 after re-evaluating the set $\mathcal{O} \setminus \mathcal{P}$ with f_M (after the model M has been re-trained). This also means using the population size qp in the EA.

The key piece of this approach is the selection criterion (or criteria) used to determine which individuals from set $\mathcal{O}$ should be used in the following generation of the algorithm. There are a number of possibilities, let us discuss the most common.

An obvious choice is selecting the best individuals based on the fitness value. This results in the region of the optimum being sampled thoroughly, which helps finding the true optimum. On the other hand, the regions far from the current optimum are neglected and a possible better optimum can be missed. To sample the areas of the fitness landscape that were not explored yet, space-filling criteria are used, either alone or in combination with the best fitness selection or other criteria.

All the previous criteria have the fact that they are concerned with the optimization itself in common. A different approach is to use the information about the model, most importantly its accuracy, to decide which points of the input space to evaluate with the true fitness function in order to most improve it. This approach is sometimes called active learning.

2.3 Active learning

Active learning is an approach that tries to maximize the amount of insight about the modeled function gained from its evaluation while minimizing the number of evaluations necessary. The methods are used in the general field of surrogate modeling as an efficient adaptive sampling strategy. The terms adaptive sampling and active learning are often used interchangeably. We will use the term active learning for the methods based on the characteristics of the surrogate model itself, such as accuracy, with the goal of minimizing the model prediction error either globally or, more importantly, in the area of the input space the EA is exploring.

The active learning methods are most often based on the local model prediction error, such as cross-validation error. Although some methods are independent of the model, for example the LOLA-Voronoi method [2], most of them depend on the model used. The kriging model used in our proposed method offers a good estimate of the local model accuracy by giving an error estimate of its prediction. It is possible to use the estimate itself as a measure of the model's confidence in the prediction, or base a more complex measure on the variance estimate. The measures that were tested for use in our method will be described in detail in section 4.1.

3 Kriging meta-models

The kriging method is an interpolation method originating in geostatistics [9], based on modeling the function as a realization of a stochastic process [11].

In the ordinary kriging, which we use, the function is modeled as a realization of a stochastic process

$$Y(\mathbf{x}) = \mu_0 + Z(\mathbf{x}) \tag{1}$$

where $Z(\mathbf{x})$ is a stochastic process with mean 0 and covariance function $\sigma^2\psi$ given by

$$cov\{Y(\mathbf{x}+\mathbf{h}), Y(\mathbf{x})\} = \sigma^2\psi(\mathbf{h}), \tag{2}$$

where σ^2 is the process variance for all $\mathbf{x}$. The correlation function $\psi(\mathbf{h})$ is then assumed to have the form

$$\psi(\mathbf{h}) = \exp\left[-\sum_{l=1}^{d} \theta_l |\mathbf{h}_l|^{p_l}\right], \tag{3}$$

where $\theta_l, l = 1, \ldots, d$, where d is the number of dimensions, are the correlation parameters. The correlation function depends on the difference of the two points and has the intuitive property of being equal to 1 if $\mathbf{h} = \mathbf{0}$ and tending to 0 when $\mathbf{h} \to \infty$. The θ_l parameters determine how fast the correlation tends to zero in each coordinate direction and the p_l determines the smoothness of the function.

The ordinary kriging predictor based on n sample points $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ with values $\mathbf{y} = (y_1, \ldots, y_n)'$ is then given by

$$\hat{y}(\mathbf{x}) = \hat{\mu_0} + \psi(\mathbf{x})'\mathbf{\Psi}^{-1}(\mathbf{y} - \hat{\mu_0}\mathbf{1}), \tag{4}$$

where $\psi(\mathbf{x})' = (\psi(\mathbf{x}-\mathbf{x}_1), \ldots, \psi(\mathbf{x}-\mathbf{x}_n))$, $\mathbf{\Psi}$ is an $n \times n$ matrix with elements $\psi(\mathbf{x}_i - \mathbf{x}_j)$, and

$$\hat{\mu_0} = \frac{\mathbf{1}'\mathbf{\Psi}^{-1}\mathbf{y}}{\mathbf{1}'\mathbf{\Psi}^{-1}\mathbf{1}}. \tag{5}$$

An important feature of the kriging model is that apart from the prediction value it can estimate the prediction error as well. The kriging predictor error in point $\mathbf{x}$ is given by

$$s^2(\mathbf{x}) = \hat{\sigma}^2\left[1 - \psi'\mathbf{\Psi}^{-1}\psi + \frac{(1-\psi'\mathbf{\Psi}^{-1}\psi)^2}{\mathbf{1}'\mathbf{\Psi}^{-1}\mathbf{1}}\right] \tag{6}$$

where the kriging variance is estimated as

$$\hat{\sigma}^2 = \frac{(\mathbf{y} - \hat{\mu_0}\mathbf{1})\mathbf{\Psi}^{-1}(\mathbf{y} - \hat{\mu_0}\mathbf{1})}{n}. \tag{7}$$

The parameters θ_l and p_l can be estimated by maximizing the likelihood function of the observed data.

For the derivation of the equations 4 - 7 as well as the MLE estimation of the parameters the reader may consult a standard stochastic process based derivation by Sacks et al. in [11] or a different approach given by Jones in [7].

4 Method description

In this section we will describe the proposed method for kriging-model-assisted evolutionary optimization with batch fitness evaluation. Our main goal was to decouple the true fitness function sampling from the EA iterations based on an assumption that requiring a specific number of true fitness evaluations in every generations of the EA forces unnecessary sampling.

In the generation-based approach, some of the points may be unnecessary to evaluate, as they will not bring any new information to the surrogate model. The individual-based approach is better suited for the task, as it chooses those points from each generation, which are estimated to be the most valuable for the model. There is still the problem of performing a given number of evaluations in every generation, although there might not be enough valuable points to select from.

The method we propose achieves the desired decoupling by introducing an evaluation queue. The evolutionary algorithm uses the model prediction at all times and when a point, in which the model's confidence in its prediction is low, is encountered, it is added to the evaluation queue. Once there are enough points in the queue, all the points in it are evaluated and the model is re-trained using the results. The optimization takes the following course.

1. Initial set $\mathcal{S}$ of b samples is selected using a chosen initial design strategy and evaluated using the true fitness function f_t.
2. An initial kriging model M is trained using pairs $(\mathbf{x}, f_t(\mathbf{x})) \in \mathcal{S}$.
3. The evolutionary algorithm is started, with the model prediction f_M as the fitness function.
4. For every prediction $f_M(\mathbf{x}) = \hat{y}_M(\mathbf{x})$, an estimated improvement measure $c(s_M^2(\mathbf{x}))$ is computed from the error estimate $s_M^2(\mathbf{x})$. If $c(s_M^2(\mathbf{x})) > t$, an improvement threshold, the point is added to the evaluation queue $\mathcal{Q}$.
5. If the queue size $|\mathcal{Q}| \geq b$, the batch size, all points $\mathbf{x} \in \mathcal{Q}$ are evaluated, the set $\mathcal{S}$ is replaced by $\mathcal{S} \cup \{(\mathbf{x}, f_t(\mathbf{x})\}$ and the EA is resumed.
6. Steps 4 and 5 are repeated until the goal is reached, or a stall condition is fulfilled.

The b and t parameters, as well as the function $c(s^2)$, are chosen before running the optimization. Note that the evaluation in step 5 can be performed either immediately, i.e. online, or offline. In offline evaluation, after filling the evaluation queue, the EA is stopped when the current iteration is finished and the control is returned to the user. After obtaining the fitness values for the samples in the sample queue (e.g. by performing an experiment), the user can manually add the samples and resume the EA from the last generation.

While the choice of the parameters will be discussed in section 5, let us introduce three different measures of estimated improvement in the model prediction $c(s^2(\mathbf{x}))$ which we tested – the standard deviation, the probability of improvement and the expected improvement.

4.1 Measures of estimated improvement

To estimate the improvement, which evaluation of a given point will bring, we can use several measures. The three measures introduced here are all based on the prediction error estimate of the kriging model. The goal of these measures is to prefer the points that help improve the model in regions explored by the EA.

Each of the measure's results for a given point are compared with a threshold and when the estimated improvement is above the threshold, the point is evaluated using the true fitness function.

Standard deviation (STD) is the simplest measure we tested. It is computed directly from the error as its square root

$$STD(x) = \sqrt{\hat{s_M^2}(\mathbf{x})}. \tag{8}$$

The STD captures only the model's estimate of the error of its own prediction (based on the distance from the known samples). As such, it does not take into account the value of the prediction itself and can be considered a measure of the model accuracy.

Probability of improvement (POI) [7] uses the fact, that the kriging prediction is a Gaussian process and the prediction in a single point is therefore a normally distributed random variable $Y(\mathbf{x})$ with a mean and variance given by the kriging predictor. If we choose a target T (based on the goal of the optimization), we can estimate the probability that a given point will have a value $y(\mathbf{x}) \leq T$ as a probability that $Y(\mathbf{x}) \leq T$. The probability of improvement is therefore defined as

$$POI(x) = \Phi\left(\frac{T - \hat{y}_M(\mathbf{x})}{s_M^2(\mathbf{x})}\right), \tag{9}$$

where Φ is the cumulative distribution function of the standard normal distribution. As opposed to the STD, the POI takes into account the prediction mean (value) as well as its variance (error estimate). The area of the current optimum is therefore preferred over the rest of the input space. When the area of the current optimum is sampled enough, the variance becomes very small and the term $\frac{T-\hat{y}_M(x)}{s_M^2(x)}$ becomes extremely negative, encouraging the sampling of less explored areas.

Expected improvement (EI) [7, 8] is based on estimating, as the name suggests, the improvement we expect to achieve over the current minimum $f_{\min}$, if a given point is evaluated. As before, we assume the model prediction in point $\mathbf{x}$ to be a normally distributed random variable $Y(\mathbf{x})$ with a mean and variance given by the kriging predictor. We achieve an improvement I over $f_{\min}$ if $Y(\mathbf{x}) = f_{\min} - I$. As shown in [7] the expected value of I can be obtained using the likelihood of achieving the improvement

$$\frac{1}{\sqrt{2\pi}s_M^2(\mathbf{x})} \int_{I=0}^{I=\infty} \exp\left[-\frac{(f_{\min} - I - \hat{y}_M(\mathbf{x})^2}{2s_M^2(\mathbf{x})}\right] \quad (10)$$

Expected improvement is the expected value of the improvement found by integrating over this density. The resulting measure EI is defined as

$$EI(\mathbf{x}) = E(I) = s_M^2(\mathbf{x})[u\Phi(u) + \phi(u)], \phi(u)], \quad (11)$$

where

$$u = \frac{f_{\min} - \hat{y}_M(\mathbf{x})}{s_M^2(\mathbf{x})} \quad (12)$$

and Φ and ϕ are the cumulative distribution function and the probability distribution functions of the normal distribution respectively. The expected improvement has an important advantage over the POI: it does not require a preset target T, which can be detrimental to the POI's successful sample selection when set too high or too low.

All three measures have an important weakness of being based on the model prediction. If the modeled function is deceptive, the model can be very inaccurate while estimating a low variance. A good initial sampling of the fitness function is therefore very important. The success of the whole method is dependent on the model's ability to capture the response surface correctly and thus on the function itself.

5 Results and discussion

The proposed method was tested using simulations on three standard benchmark functions. We studied the model evolution during the course of the optimization, the effect of the parameters and also investigated the optimal choice of batch size for problems where an upfront choice is possible. In this section we discuss the tests performed and their results.

For testing, we used the genetic algorithm implementation from the global optimization toolbox for the Matlab environment and the implementation of an ordinary kriging model from the SUMO Toolbox [4]. The parameters of the supporting methods, e.g. the genetic algorithm itself, were kept on their default values provided by the implementation.

Because the EA itself is not deterministic, each test was performed 20 times and the results we present are statistical measures of this sample. As a performance measure we use the number of true fitness evaluations used to reach a set goal in all tests. The main reason to use this measure is that in model-assisted optimization the computational cost of everything except the true fitness evaluation is minimal in comparison. We also track the proportion of the 20 runs that reached the goal before various limits (time, stall, etc.) took effect.

5.1 Benchmark functions

Since the evolutionary algorithms and optimization heuristics in general are often used on black-box optimization, where the properties of the objective function are unknown, it is not straightforward to asses their quality on real world problems. It has therefore become a standard practice to test optimization algorithms and their modifications on specially designed testing problems.

These benchmark functions are explicitly defined and their properties and optima are known. They are often designed to exploit typical weaknesses of optimization algorithms in finding the global optimum. We used three functions found in literature [10]. Although we performed our tests in two dimensions we give general multi-dimensional definitions of the functions.

First of the functions used is the De Jong's function. It is one of the simplest benchmarks, it is continuous, convex and unimodal and is defined as

$$f(\mathbf{x}) = \sum_{i=1}^{n} x_i^2 \quad (13)$$

The domain is restricted to a hypercube $-10 \leq x_i \leq 10, i = 1, \ldots, n$. The function has one global optimum $f(\mathbf{x}) = 0$ in point $\mathbf{x} = \mathbf{0}$. The De Jong's function was primarily used as a proof of concept test.

As a second benchmark, we used the Rosenbrock's function, also called Rosenbrock's valley. The global optimum is inside a long parabolic shaped valley, which is easy to find. Finding the global optimum in

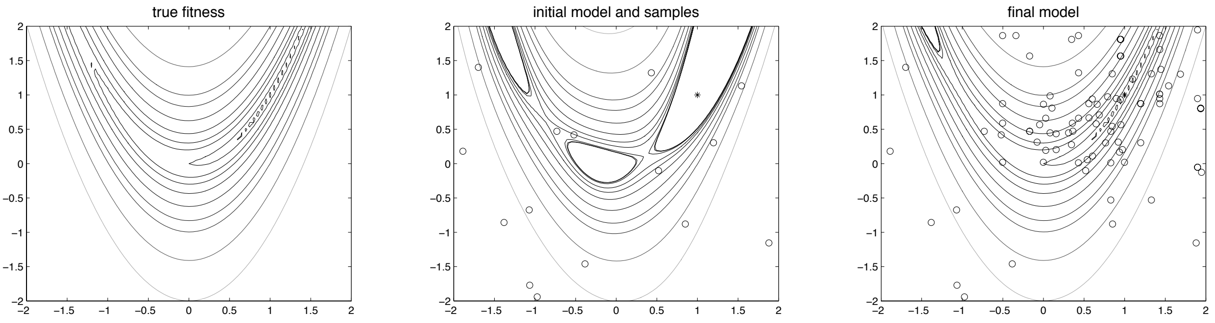


Fig. 1. The original fitness function, the initial model and the final model.

that valley however is difficult [10]. The function has the following definition

$$f(\mathbf{x}) = \sum_{i=1}^{n} [100(x_{i+1} + x_i^2)^2 + (1 - x_i)^2] \tag{14}$$

The domain of the function is restricted to a hypercube $-2 \leq x_i \leq 2, i = 1, \ldots, n$. It has one global optimum $f(\mathbf{x}) = 0$ in $\mathbf{x} = \mathbf{1}$.

Finally, the third function used as a benchmark is the Rastrigin's function. It is based on the De Jong's function with addition of cosine modulation, which produces a high number of regularly distributed local minima and makes the function highly multimodal. The function is defined as

$$f(\mathbf{x}) = 10n + \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i)] \tag{15}$$

The domain is restricted to $-5 \leq x_i \leq -5$, $i = 1, \ldots, n$. The global optimum $f(\mathbf{x}) = 0$ is in $\mathbf{x} = \mathbf{1}$.

5.2 Model evolution

As the basic illustration of how the model evolves during the course of the EA, let us consider an example test run using the Rosenbrock's function. For this experiment we set the batch size of 15, used the STD measure of estimated improvement with a threshold of 0.001 and set the target fitness value of 0.001 as well. The target was reached at the point (0.9909, 0.9824) using 90 true fitness evaluations. A genetic algorithm without a surrogate model needed approximately 3000 evaluations to reach the goal in several test runs.

The model evolution is shown in figure 1. The true fitness function is shown on the left, the initial model is in the middle and the final model on the right. The points where the true fitness function was sampled are denoted with circles an the optimum is marked with a star.

function	ev (1q)	ev (med)	ev (3q)	goal	reached
De Jong	60	60	120	0.01	1
Rosenbrock	60	125	310	0.1	1
Rastrigin	260	370	580	0.1	0.85

Table 1. GA performance on benchmark functions without a model.

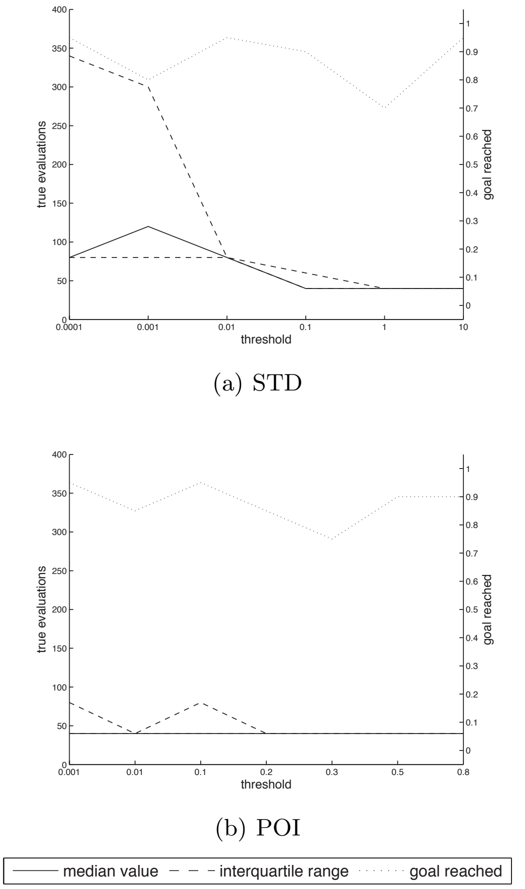
5.3 Measures of estimated improvement comparison

In order to compare the measures of estimated improvement, we performed simulations on each benchmark using each improvement estimate measure with different values of the threshold. The batch size was set to 40 – generally found to be the ideal batch size – for these experiments. For comparison, we also performed tests with the standard genetic algorithm without a model. Results of these simulations are shown in the table 1.

The De Jong's function proved to be simple to optimize and the threshold setting did not have almost any effect. Only when using the standard deviation, setting the threshold too low lead to an increase in the number of evaluations, as too many points were evaluated, although the model prediction in those points was accurate enough.

The same is true for the STD measure used on the Rosenbrock's function, where setting the threshold too low leads to a big increase in variance of the results. Interestingly, setting the threshold too high leads to a decrease in the number of evaluations, but also in the success rate of reaching the goal. The POI and EI are more stable in terms of true fitness evaluations, but have worse overall success rate. The results are shown in figure 2

The Rastrigin's function proved difficult to optimize. This is probably due to the locality of the kriging model and the high number of local minima of the function. Overall the STD measure is the most suc-



(a) STD

(b) POI

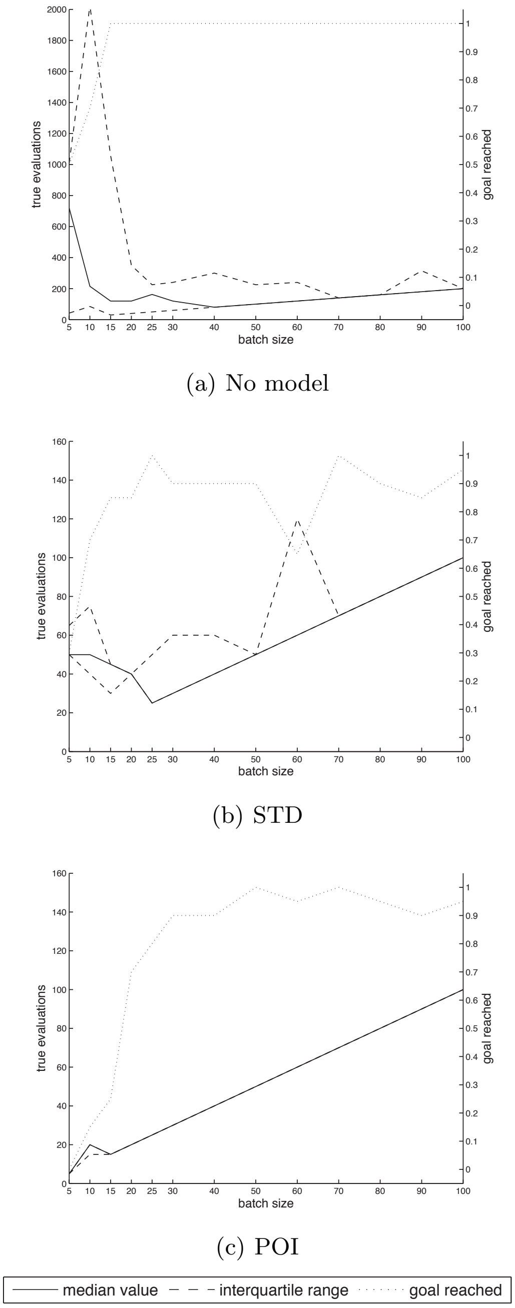
Fig. 2. STD measure on the Rosenbrock's function - true fitness evaluations and proportion of runs reaching goal.

cessful. POI and EI lead to bad sampling of the model and failure to reach the optimum.

An interesting general result is that the more complex measures of estimated improvement perform worse than the simple standard deviation estimate. This indicates that the goal of active learning selection criteria in the evolutionary optimization should be the best possible sampling for overall model accuracy, as opposed to trying to improve the accuracy in the best regions of the input space. Both the POI and EI are design to select next best points to reach the optimum. Since in our case, this is handled by the EA itself, the measures bring an unnecessary noise to the estimate of the model accuracy. The results also show that the best measure selection is dependent on the optimized function.

5.4 Batch size

In order to study the batch size effect on the optimization, a number of experiments were performed with different batch sizes. The only option to achieve a given batch size is to set the population size in a standard GA, in our method however, the settings are independent so a population size of 30, which proved efficient, was used in all of the tests.



(a) No model

(b) STD

(c) POI

Fig. 3. Batch size effect on Rosenbrock's function optimization - true fitness evaluations and proportion of runs raching the goal.

The results on the De Jong's functions show that apart from small batch sizes (up to 10), the optimization is successful in all runs. Our method helps stabilize the EA for small batch sizes and for batch sizes above 15 the algorithm finds the optimum using a single batch. For a standard GA this strong dependence arises for batch sizes above 40 and the algorithm reaches the goal in the second generation, evaluating twice as many points.

For the Rosenbrock's function we get the intuitive result that setting the batch size too low leads to more evaluations or a failure to reach the goal, while large batch size do not improve the results and waste true fitness evaluations. For this function the POI proved to be the most efficient measure. The comparison is shown in figure 3. Overall the method reduces the number of true evaluations from hundreds to tens for the Rosenbrock's function, while slightly reducing the success rate of the computation.

The Rastrigin's function proved difficult to optimize even without a surrogate model. With the model, the STD achieved the best results reducing the number of true fitness evaluations approximately three times in the area of the highest success rate with batch size of 70. The other two measures were ineffective. We attribute the method's difficulty optimizing the Rastrigin's function to the fact that the kriging model is local and thus it requires a large number of samples to capture the function's complicated behavior in the whole input space. When the initial sampling is misleading, which is more likely for the Rastrigin's function, both the model prediction and estimated improvement are wrong.

The results suggest that best batch size and best estimated improvement measure are highly problem-dependent. The proposed method is also very sensitive to good initial sample selection, which is the most usual reason for it to fail to find the optimum. The experimental results support the intuition that batches too small are bad for the initial sampling of the model and batches too large slow down the model improvement by evaluating points that it would not be necessary to evaluate with smaller batches. This suggests using a larger initial sample and a small batch for the rest of the optimization.

6 Conclusions

In this paper we presented a method for model-assisted evolutionary optimization with a fixed batch size requirement. To decouple the sampling from the EA iterations and support an individual-based approach while keeping a fixed evaluation batch size, the method uses an evaluation queue. The candidates for true fitness evaluations are selected by an active learning method using a measure of estimated improvement of the model quality based on the model prediction error estimate.

The results suggest using simple methods for improvement estimate in active learning, which only capture information about the model accuracy improvement expected by sampling a given point. In the experiments with the batch size we found that small batch sizes perform better when the objective function is simple, while causing bad initial sampling of more complex functions, suggesting using a larger initial sample. The future development of this work should include experiments using different batch sizes for initial sampling and comparison of the method with other ways of employing a surrogate model in the optimization as well as other model-assisted optimization methods.

The method brings promising results, reducing the number of true fitness evaluations to a large degree for some of the benchmark functions. On the other hand, its success is highly dependent on the optimized function and its initial sampling.

References

1. M. Baerns, M. Holeňa: *Combinatorial development of solid catalytic materials: design of high-throughput experiments, data analysis, data mining.* Catalytic Science Series. Imperial College Press, 2009.
2. K. Crombecq, L. De Tommasi, D. Gorissen, T. Dhaene: *A novel sequential design strategy for global surrogate modeling.* In Winter Simulation Conference, WSC '09, Winter Simulation Conference, 2009, 731–742.
3. D. Gorissen: *Grid-enabled adaptive surrogate modeling for computer aided engineering.* PhD Thesis, Ghent University, University of Antwerp, 2009.
4. D. Gorissen, I. Couckuyt, P. Demeester, T. Dhaene, K. Crombecq: *A surrogate modeling and adaptive sampling toolbox for computer based design.* The Journal of Machine Learning Research 11, 2010, 2051–2055.
5. L. Gräning, Y. Jin, B. Sendhoff: *Efficient evolutionary optimization using individual-based evolution control and neural networks: A comparative study.* In ESANN, 2005, 273–278.
6. Y. Jin, M. Olhofer, B. Sendhoff: *Managing approximate models in evolutionary aerodynamic design optimization.* In Evolutionary Computation, 2001. Proceedings of the 2001 Congress on, vol. 1, IEEE, 2001, 592–599.
7. D.R. Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21:345–383, 2001.
8. D. R. Jones, M. Schonlau, W.J. Welch: *Efficient global optimization of expensive black-box functions.* Journal of Global Optimization 13, 1998, 455–492.
9. G. Matheron: *Principles of geostatistics.* Economic Geology 58(8), 1963, 1246–1266.
10. M. Molga, C. Smutnicki: *Test functions for optimization needs.* Test Functions for Optimization Needs, 2005.
11. J. Sacks, W. J. Welch, T. J. Mitchell, H. P. Wynn: *Design and analysis of computer experiments.* Statistical Science 4(4), 1989, 409–423.

Aligning sequences with repetitive motifs[⋆]

Peter Kováč, Broňa Brejová, and Tomáš Vinař

Faculty of Mathematics, Physics, and Informatics, Comenius University in Bratislava,
Mlynská dolina, 842 48 Bratislava, Slovakia
kovac.peter@fotopriestor.sk, {vinar,brejova}@fmph.uniba.sk

Abstract. *Pairwise sequence alignment is among the most intensively studied problems in computational biology. We present a method for alignment of two sequences containing repetitive motifs. This is motivated by biological studies of proteins with zinc finger domain, an important group of regulatory proteins. Due to their evolutionary history, sequences of these proteins contain a variable number of different zinc fingers (short subsequences with specific symbols at each position).*

Our algorithm uses two types of hidden Markov models (HMM): pair HMMs and profile HMMs. Profile HMMs describe the structure of sequence motifs. Pair HMMs assign a probability to alignment of two motifs. Combination of the these two types of models yields an algorithm that uses different score when aligning conserved vs. variable motif residues. The dynamic programming algorithm that computes the motif alignments is based on the well known Viterbi algorithm. We evaluated our model on sequences of zinc finger proteins and compared it with existing alternatives.

1 Introduction

Pairwise sequence alignment is one of the most studied problems in bioinformatics. We will concentrate on alignment of protein sequences, where a protein can be represented as a string over the alphabet of 20 different amino acids. During the evolution, particular amino acids in a protein can be substituted by another amino acid, or even get inserted or deleted. The goal of sequence alignment is to compare two proteins, quantify their sequence similarity, and to identify pairs of amino acids that have likely evolved from the same amino acid in the common ancestor. Over the years, multitude of variations of this problem have been introduced and many practical software tools were developed.

Our work is motivated by the study of zinc finger proteins. These proteins contain a variable number of up to 40 zinc finger domains [18]. Zinc finger domain is a stretch of approximately 28 amino acids, the purpose of which is to bind DNA at specific places. Comparison of zinc fingers form different proteins reveals that some positions are very conserved due to their importance in assuming desired function, while other positions are highly variable, since they distinguish specific DNA sequences where individual zinc fingers bind (Figure 1).

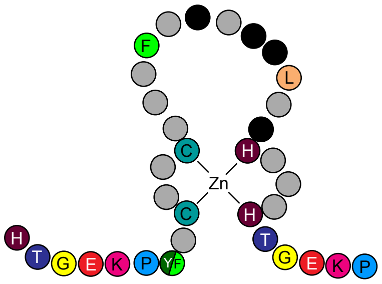


Fig. 1. The structure of a zinc-finger. Highly variable sites are marked with black color. The most conserved amino acids are the four involved in binding the zinc ion [14].

We will focus our attention on the KRAB-ZNF proteins that have a region encoding one or more Krüppel-associated box domains (KRAB, [2]) followed by a zinc finger region (Fig. 2). The human genome encodes more that 600 of proteins from this family, and a lot of effort is dedicated to building and maintaining their catalogues [9], [11], [3]. Complicated repetitive structure of these genes is a result of a dynamic evolutionary history, full of sequence duplications [7, 12], and many mutations which help to gain new functions for duplicated copies.

The repetitive nature of zinc finger protein sequences complicates their sequence alignment. Traditional alignment methods based purely on sequence similarity frequently misalign individual zinc fingers, or even align a single zinc finger in one sequence to parts of several different zinc fingers in the other sequence. Consequently, many studies of these proteins limit their analyses and infer conclusions based only on the the KRAB domains or sequences before the zinc finger region (e.g. [14], [7]), or dispute the relevance of standard methods applied to genes with high variance in the number of fingers [16].

In this work, we develop a new method for aligning sequences with repetitive motifs, such as zinc finger proteins. To overcome the problems outlined above, we combine the strength of profile hidden Markov models

[⋆] This work was supported by the European Community FP7 Marie Curie grants IRG-224885 to TV and IRG-231025 to BB, and by a grant from VEGA 1/1085/12.

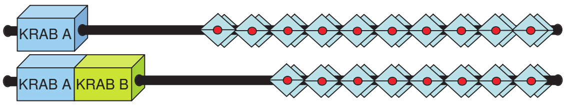


Fig. 2. Domain structure of typical KRAB-ZNF genes. The protein contains one or more KRAB domains and an array of 3 to cca. 40 zinc fingers. [18]

which are used to characterize the properties of these repetitive motifs, and pair hidden Markov models as a model of sequence alignments. We compare our work to MotifAligner that was previously used to align zinc finger proteins [11], and we find our new method to produce more accurate alignments on our testing set.

In the rest of this section, we introduce necessary background and notation and describe the MotifAligner approach to repetitive sequence alignment in more detail. In Section 2, we describe our new profile-profile-pair alignment method (PPP). We present the results of experimental comparison of PPP and MotifAligner in Section 3.

1.1 Background and notation

In this paper, we rely on several standard tools from computational biology, namely alignments, pair hidden Markov models, and profile hidden Markov models, which we briefly explain in this section.

We start by defining hidden Markov models (HMMs). An HMM is a probabilistic finite state automaton. We can use it to generate a random sequence over some alphabet as follows. We start in a designated start state B. In each step, we sample a character of the sequence from the emission probability distribution associated with the current state and then randomly change the state according to the transition probability distribution. The process ends when we reach the designated final state E.

The sequence of states visited in the individual steps is called a state path. We will denote the probability of emitting x in state v as $e_v(x)$ and the probability of transition from state v to w as $t_{v,w}$. The joint probability of emitting a sequence $x = x_1 \dots x_n$ along the state path $s = s_1 \dots s_n$ in a given HMM is

$$P(x,s) = e_{s_1}(x_1) \prod_{i=2}^{n} t_{s_{i-1}s_i} e_{s_i}(x_i).$$

A typical task solved with HMMs is to find the most probable state path that could generate a given sequence, i.e. to find $s^* = \arg\max_s P(x,s)$. This task is solved by the Viterbi algorithm based on dynamic programming [19].

The second important notion is sequence alignment. Given a set of related protein sequences, we can align them by inserting dashes to individual sequences so that they all have the same length and when we arrange them in a table, as in Figure 3, many columns contain the same or similar amino acids. Several consecutive dashes form a gap in the alignment, indicating that a part of the sequence was deleted or inserted during the evolution. The sequence alignment problem can be formulated as an optimization problem and solved by existing algorithms. For two sequences, the problem can be solved easily by Needleman-Wunsch dynamic programming algorithm [10], for multiple sequences it is NP-hard [6]. The scoring function for pairwise alignment is typically based on a substitution matrix scoring all pairs of aligned amino acids and on parameters for scoring gaps: gap opening penalty g for the first dash in a gap and gap extension penalty e for each additional gap.

```
ZNF626_4799/12   YKC--EECGKAF-NQSSILTTHERIILERN-
ZNF727_4861/2    YKC--EECGKDC--RLSDFTIQKRIHTADRS
ZXDB_644/5       YQCAFSGCKKTF-ITVSALFSHNRAHFREQE
LLNL1236_4814/2  SMC--PECSKTSATDSSCLLMHQRSHTGKRP
ZNF23_141/15     FQC--KECGKAF-HVNAHLIRHQRSHTGEKP
```

Fig. 3. Alignment of five sequences of zinc finger motifs from human proteins.

One way of systematically deriving a scoring function for pairwise alignments is to use pair HMMs [4]. These models emit two sequences simultaneously. In one step, the HMM can emit a single character in one of the sequences or in both. The later case corresponds to two symbols aligned to each other, the former to a symbol aligned to a dash. Figure 4 shows the pair HMM used in our work. The match state M emits pairs of aligned characters, state X emits characters only in the first sequence, and state Y emits characters only in the second sequence. Given two sequences, we can find the most probable state path that could generate them and this will give us an alignment of these two sequences.

To represent a typical sequence of a motif, we will use another kind of HMMs, called profile HMMs [4]. A profile HMM is typically constructed based on an alignment of several motif instances, such as the one in Figure 3. Each position of the motif is represented by one state with emission probabilities set to the observed frequencies of amino acids in the corresponding alignment column (possibly with some pseudocounts added to avoid zero probabilities). These so called match states are arranged in a chain (see Figure 5). These states used alone would generate sequences of the same length. However, real sequences may have various insertions and deletions compared to the con-

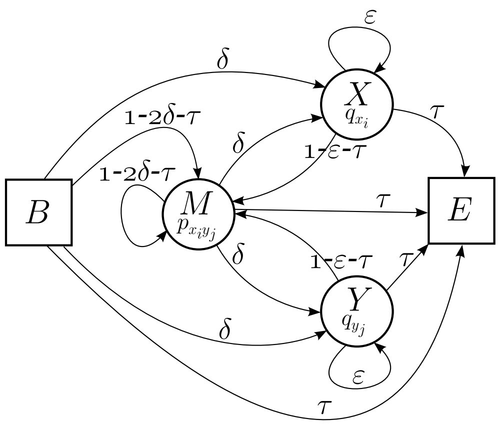


Fig. 4. A pair HMM for global alignment. Transition probabilities are defined by three parameters $\delta, \varepsilon, \tau$, emission probabilities by matrices p and q.

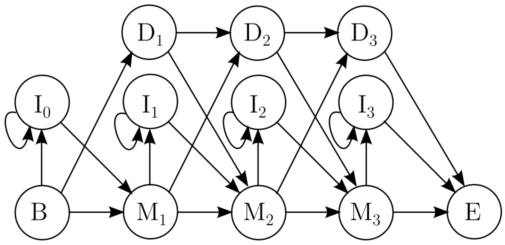


Fig. 5. Example of a profile HMM. States M_k are match states, I_k are insert states and D_k are delete states. States B, E, and $D_1 \dots D_3$ are silent, which means that they do not generate any characters.

sensus motif; these are modeled by additional insert and delete states. Given a profile HMM and a sequence, we can again find the most probable state path, which in this case gives us an alignment of the sequence to the motif represented by the profile HMM. Note, however, that the profile HMM emits only a single sequence; the motif itself is represented directly in the structure and parameters of the model.

1.2 MotifAligner approach

To obtain high quality alignments even on sequences with highly variable number of zinc finger motifs, Nowick et al. developed a pairwise alignment tool called MotifAligner [11]. To our knowledge, it is the only sequence alignment method designed specifically to align sequences with variable number of repetitive motifs. Part of our work was inspired by this algorithm.

MotifAligner first uses a profile HMM tool HMMER [5] and finds all canonical motif occurrences with statistically significant scores in both input sequences. Let $T = (t_1, \dots, t_a)$ and $U = (u_1, \dots, u_b)$ be the sequences of all motif occurrences found by HMMER in the original input sequences x and y, respectively. In the second step, MotifAligner computes scores of all gapless pairwise alignments of motifs t_k, u_ℓ, for all $1 \le k \le a$, $1 \le \ell \le b$:

$$s[t_k, u_\ell] = \sum_{i=1}^{L} S[t_{k_i}, u_{\ell_i}], \qquad (1)$$

where $S[x_i, y_j]$ is the score of aligning amino acids x_i and y_j (they use a standard BLOSUM85 substitution matrix [8]; motif occurrences are padded to have the same length).

In this way we obtain a similarity score between each pair of motif occurrences. Next MotifAligner applies the Needleman-Wunsch algorithm [10] to T and U, treating motifs as sequence symbols and using matrix s as the substitution matrix. In this way we obtain pairs of aligned zinc fingers between the two proteins.

2 Profile-profile-pair alignment

In this section, we present a new approach to alignment of sequences with repetitive motifs. We adopt an approach similar to MotifAligner, however, we change the alignment algorithm and the scoring scheme to take into account the structure of the repeated motif.

For example, the zinc-finger motif (Fig. 2) contains several highly conserved positions, among them the four amino-acids binding the zinc ion (positions 3, 7, 20, 24). These four amino acids are crucial to the function of the motif and as such should be used to anchor the whole alignment. However, the fact that these positions match in the two aligned sequences should not be very surprising and should not by itself contribute much to the resulting score. On the other hand, there are several variable positions, and the differences at these positions will be very informative of the evolutionary distance.

To take these issues into account, we have developed a new *profile-profile-pair* method (PPP) for

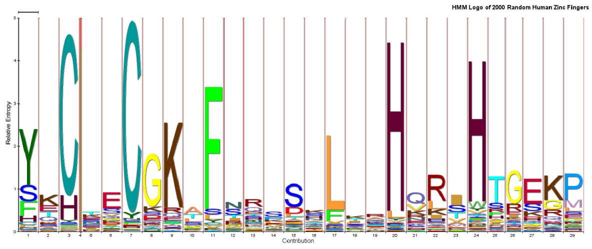


Fig. 6. The profile HMM of random 2000 human zinc fingers from the complete dataset, viewed as a HMM logo [15].

alignment of individual motifs. The method uses a combination of two profile HMMs and a pair HMM for sequence alignment and aligns the two sequences by finding the best possible path through all three models simultaneously. To align the complete protein sequences containing these repeating motifs, we first align each possible pair of motifs through PPP, compute their similarity score, and use a modification of a traditional global alignment algorithm, now operating on individual motif occurrences as a unit. We describe the details of the method in the remainder of this section.

2.1 Pairwise alignment of individual motifs

The input to PPP consists of two instances of the repeating motif $x = x_1 \dots x_{L_x}$ and $y = y_1 \dots y_{L_y}$, a profile HMM encoding the same motif, and a pair HMM characterizing the properties of a typical alignment. Our goal is to align both x and y to a separate copy of profile HMM and at the same time, use the pair HMM as a glue.

In particular, we are simultaneously seeking the three paths through the three HMMs that satisfy the following constraints:

Constraint 1 (Profile match states constraint) *If x_i and y_j are emitted by the same match state M_k in their profile models then the pair model has emit x_i and y_j together in the match state M.*

Constraint 2 (Pair match state constraint) *If the pair model emits x_i and y_j together in the match state M then both profile models emit x_i and y_j in the same match state M_k or in the same insert state I_k.*

In other words, if the pair model is in the state X or Y (which is interpreted as a gap in one of the sequences), the two profile models should not be in the same match state: symbols belonging to the same consensus column should be aligned. However, if both profile models are in the same insert state they can either be evolutionarily related, in which case they should be aligned using M state of the pair model, or they could have been inserted in the sequence independently, which would correspond to using X and Y states of the pair model. Constraint 2 also implies, that if the profile models are neither in the same match state M_k nor in the same insert state I_k (i.e. either are in completely different columns or in the same column k, but different states M_k and I_k), which means that the symbols being emitted are unrelated, then the pair model should not be in the match state. These constraints thus ensure that the sequence and the profile alignment can be interpreted in a consistent manner.

Thus our goal is to compute three paths s_p^*, s_x^*, s_y^* through the pair HMM and the two profile HMMs that would satisfy our constraints and the product of joint probabilities implied by all three models would be maximized:

$$(s_p^*, s_x^*, s_y^*) = \arg \max_{\substack{\text{valid} \\ s_p, s_x, s_y}} \text{score}(x, y, s_p, s_x, s_y), \qquad (2)$$

where $\text{score}(x, y, s_p, s_x, s_y) = P_{\text{pair}}(x, y, s_p) \cdot P_{\text{profile}}(x, s_x) \cdot P_{\text{profile}}(y, s_y)$, where $P_{\text{pair}}(x, y, s)$ is the joint probability of the state path s aligning sequences x and y in the pair HMM and $P_{\text{profile}}(x, s)$ is the joint probability of the state path s and sequence x in the profile HMM.

We obtain an optimal solution using dynamic programming similar to the Viterbi algorithm used to compute the most probable state paths in individual HMMs. Let $S = (S_p, S_x, S_y)$ be the triplet of states of pair and x-profile and y-profile models satisfying our conditions. We denote $V[S_p, S_x, S_y, i, j]$ the score of the highest scoring state path combination ending with the triplet S and covering the prefixes $x_1 \dots x_i$, $y_1 \dots y_j$ of the two sequences.

The computation of $V[S_p, S_x, S_y, i, j]$ depends on the types of states S_p, S_x, S_y. For example, if S_p is the match state M of the pair HMM and S_x is the match state M_k of the profile HMM, then according to our constraints S_y must be the same match state M_k and we have the following recurrence:

$$V[M, M_k, M_k, i, j] = e_M(x_i, y_j) e_{M_k}(x_i) e_{M_k}(y_j) \cdot$$
$$\max \begin{cases} t_{MM} t_{M_\ell M_k} t_{M_\ell M_k} \cdot V[M, M_\ell, M_\ell, i-1, j-1] \\ \qquad \text{for } 0 \le \ell < k \\ t_{MM} t_{I_{k-1} M_k} t_{I_{k-1} M_k} \cdot V[M, I_{k-1}, I_{k-1}, i-1, j-1] \\ t_{XM} t_{M_\ell M_k} t_{M_n M_k} \cdot V[X, M_\ell, M_n, i-1, j-1] \\ \qquad \text{for } 0 \le \ell, n < k, n \ne \ell \\ t_{XM} t_{M_\ell M_k} t_{I_{k-1} M_k} \cdot V[X, M_\ell, I_{k-1}, i-1, j-1] \\ \qquad \text{for } 0 \le \ell < k \\ t_{XM} t_{I_{k-1} M_k} t_{M_\ell M_k} \cdot V[X, I_{k-1}, M_\ell, i-1, j-1] \\ \qquad \text{for } 0 \le \ell < k \\ t_{XM} t_{I_{k-1} M_k} t_{I_{k-1} M_k} \cdot V[X, I_{k-1}, I_{k-1}, i-1, j-1] \\ t_{YM} t_{M_\ell M_k} t_{M_n M_k} \cdot V[Y, M_\ell, M_n, i-1, j-1] \\ \qquad \text{for } 0 \le \ell, n < k, n \ne \ell \\ t_{YM} t_{M_\ell M_k} t_{I_{k-1} M_k} \cdot V[Y, M_\ell, I_{k-1}, i-1, j-1] \\ \qquad \text{for } 0 \le \ell < k \\ t_{YM} t_{I_{k-1} M_k} t_{M_\ell M_k} \cdot V[Y, I_{k-1}, M_\ell, i-1, j-1] \\ \qquad \text{for } 0 \le \ell < k \\ t_{YM} t_{I_{k-1} M_k} t_{I_{k-1} M_k} \cdot V[Y, I_{k-1}, I_{k-1}, i-1, j-1] \end{cases}$$

The value at $V[M, M_k, M_k, i, j]$ has to include the emission probabilities of x_i and y_j in all three models. Then we take a maximum over all choices of previous cells from which the current cell value could be computed. Every value considered in the maximum is the product of the value of the predecessor cell and transition probabilities in all three models. All the other cases can be derived analogously; we omit the derivations due to the space constraints.

Every time we compute a value for any cell, we keep a pointer to the cell from which the value was derived. We use those pointers later to trace back the resulting state paths. Of particular importance is the path in the pair model, since it defines the alignment of x and y. For each of the resulting state path, we also compute its joint probability its respective model, obtaining values $P_{\text{pair}}(x, y, s^*_p)$, $P_{\text{profile}}(x, s^*_x)$, and $P_{\text{profile}}(y, s^*_y)$.

2.2 Alignment of complete motif arrays

We use the same procedure as MotifAligner for alignment of complete motif arrays. We compute all pairwise alignments of individual motifs, where the score of a pairwise motif alignment is based on joint probabilities of motif sequences and state paths in all three models as described below. Since we perform the motif alignment for all pairs of motifs and assign a score to each such alignment, we get a scoring system similar to a scoring matrix. Treating motifs as symbols and using this scoring matrix, we obtain the full alignment of input motif arrays using Needleman-Wunsch algorithm.

More formally, for motif arrays $A_x = (x_1, \ldots, x_n)$ and $A_y = (y_1, \ldots y_m)$, we calculate $n \times m$ matrix S, where

$$S(x_i, y_j) = \ln \frac{P_{\text{pair}}(x_i, y_j, s^*_{p,i,j})}{P_{\text{profile}}(x_i, s^*_{x,i,j}) P_{\text{profile}}(y_j, s^*_{y,i,j})}, \quad (3)$$

where $s^*_{p,i,j}$, $s^*_{x,i,j}$ and $s^*_{y,i,j}$ are the three state paths computed when aligning motifs x_i, y_j by PPP. This score compares the hypothesis that the two motif sequences are related (given by probability from the pair HMM) to the hypothesis that these are simply two independent sequences following the same profile (as determined by scores from the two profile HMMs).

2.3 Algorithm complexity

The time complexity of the PPP algorithm on two motif arrays with $O(n)$ motifs, each of length $O(m)$ is $O(n^2 m^6)$. There are $O(n^2)$ individual motif alignments. The time needed to compute one such alignment is $O(L_x L_y L^4)$, where L_x, L_y are the lengths of motifs and L is the number of columns in the profile HMM. This follows from the observation that in the recurrent step of individual motif alignment we fill $3 \times L \times L \times L_x \times L_y$ matrix, and time required to compute each cell is at most $O(L^2)$, the upper bound on the number of values considered in the recurrence. Typically, the number of columns in the profile HMM and the length of motifs is almost the same, so we can say that $L_x, L_y, L = O(m)$ and hence the time required to compute the alignment of one motif pair is $O(m^6)$. From the same observation, one can easily see that the space complexity is $O(n^2 + m^4)$. The running time and memory is practical, since values of n and m tend to be small in real proteins (for zinc-finger arrays, both n and m are less than 30).

	Number of		Finger Motifs		
Genome	Genes	Variants	Total	Average	Median
hg19	612	1071	13363	12.48	12
mm9	302	513	5226	10.19	10
canFam2	477	828	9259	11.18	11
rheMac2	578	1010	12143	12.02	12

Table 1. The complete dataset, based on genes from the whole human genome. One gene can have multiple variants that differ in organization of zinc fingers.

3 Experiments and evaluation

Gold standard data set. We evaluated our approach on human zinc-finger genes and their counterparts in related species macaque, mouse, and dog. We downloaded the set of annotations of KRAB zinc finger genes from the Human KZNF Catalog [9] and remapped the annotation to the current human genome assembly hg19 using *liftOver* tool. To obtain the sequences of these zinc finger genes in other species, we used the whole genome alignments from the UCSC genome browser [17] as a mapping between the human (hg19) and the macaque (rheMac2), mouse (mm9), and dog (canFam2) genomes.

The resulting genomic sequences were translated into amino acid sequences and cleaned for apparent artifacts. In particular, we removed genes that contained fingers shorter than 10 amino acids. Summary statistics of the resulting dataset is shown in Table 1. Because of relatively high time complexity of the PPP algorithm, alignment of genes with high number of fingers takes a lot of time. For that reason, we prepared a subset of the complete dataset, omitting genes from human chromosome 19 and their putative orthologs in other genomes. These genes contain the highest numbers of repeating motifs (30 or more). Summary statistics for this *restricted dataset* are shown in the Table 2.

Model parameters. The emission probabilities of the pair HMM used in our experiments were based on the BLOSUM85 substitution matrix. This particular matrix was chosen in order to compare our results to MotifAligner [11]. In particular, we used the probability

	Number of		Finger Motifs		
Genome	Genes	Variants	Total	Average	Median
hg19	323	510	5249	10.29	9
mm9	201	314	2818	8.97	8
canFam2	257	406	3710	9.14	8
rheMac2	305	484	4766	9.85	9

Table 2. The restricted dataset, omitting genes from human chromosome 19 and their orthologs.

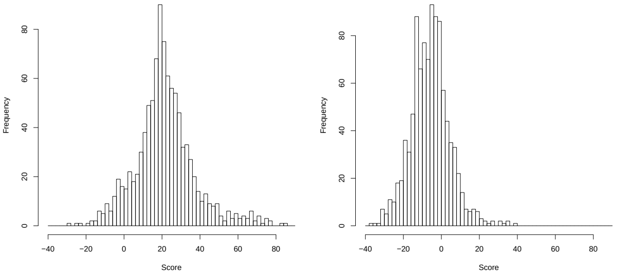


Fig. 7. Score distributions in related (left) and random (right) datasets om PPP model.

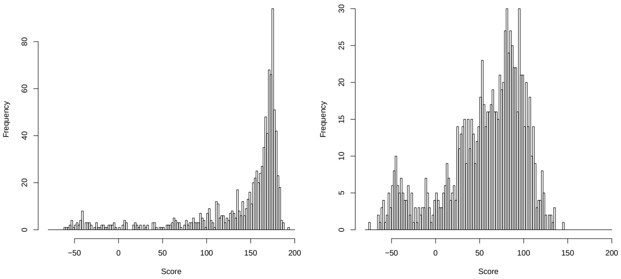


Fig. 8. The score distributions in related (left) and random (right) datasets, MotifAligner approach based on the BLOSUM85 matrix.

distributions p and q from which the matrix was derived, as supplied in EMBOSS software package [13]. The transition probability parameters (see Figure 4) were set as follows: $\tau = 0.0345$, so that the expected length of an alignment is 28, which is the length of a typical human C_2H_2 zinc finger motif; $\delta = 0.05185$ so that the expected length of a match region is 13.45, because the most variable region of a zinc finger motif spans positions 12-15; $\varepsilon = 0.4769$ so that the expected length of a gap is 1.1.

The complete parameter set of the profile model was acquired from the Pfam database entry for the ZNF C_2H_2 family [1]. The length of the profile is 23, which is shorter than a typical human zinc finger motif. The reason is that the model is based on a more diverse set of sequences from various species.

3.1 The PPP score distribution

To compare the scoring function of the PPP model with the scores used by MotifAligner, we created two sets of zinc-finger motif pairs. The *related* set contained 1000 fingers from the human genome, each paired with the corresponding finger from macaque, mouse or dog. The *random* set contained 1000 random pairs of fingers; we assume that these fingers are on average more distantly related to each other than paired fingers in the first set.

We have computed a PPP alignment of each sequence pair in both samples. The score distributions are shown in Figure 7. Both distributions resemble the normal distribution, with mean of the related set close to 20 and mean of the random set at around 5.

For comparison purposes, we have reimplemented MotifAligner algorithm as described in [11]. Figure 7 shows the score distributions of the MotifAligner approach to alignment of individual motifs, based on the BLOSUM85 substitution matrix. These score distributions do not resemble the normal distribution. In particular, the distribution for the related set has a heavy tail, which is clearly not desirable.

The important property of the scoring scheme is how well it is able to distinguish positive examples from negative ones. Figure 9 shows a ROC curve, where the related set was treated as positives and the random set as negative examples. The classification performance of the PPP is clearly better, demonstrating that our scheme is more suitable as a score for classification of paired motifs from random pairs.

3.2 Alignment accuracy

Next we use the PPP and the simpler MotifAligner method for scoring pairs of zinc fingers as building blocks in the whole motif array alignment. The Needleman-Wunsch algorithm for the whole motif array alignment has three parameters: the gap opening penalty g, the gap extension penalty e, and the substitution matrix s that scores individual motif alignments. For MotifAligner, we have used the original parameters [11], in particular the BLOSUM85 substitution matrix and the gap penalties set to $g = 84$ and $e = 75.6$. In the PPP model, the matrix s is determined by the equation 3, and we have tested several different settings of the parameters g and e.

We carried out three tests. In the first one, we aligned all zinc finger arrays of orthologous proteins in the complete dataset. The second and the third experiments simulated a loss of fingers during the evolution – we created two artificial datasets with 1/5 and 1/3 of the total number of fingers removed in each zinc finger array in all four genomes, and we aligned the original human dataset with the four reduced sets.

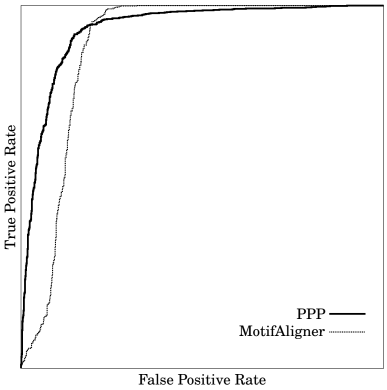


Fig. 9. ROC curve for related (positive) and random (negative) datasets.

Dataset	Program	Aligned arrays	Misaligned motifs
Complete, Unchanged	MotifAligner	2161	234
Complete, Unchanged	PPP^1	2161	178
Complete, Unchanged	PPP^2	2161	331
Restricted, 1/5 Loss	MotifAligner	1609	149
Restricted, 1/5 Loss	PPP^1	1609	139
Restricted, 1/5 Loss	PPP^2	1609	142
Restricted, 1/3 Loss	MotifAligner	1651	169
Restricted, 1/3 Loss	PPP^1	1651	254
Restricted, 1/3 Loss	PPP^2	1651	252

Table 3. The comparison of MotifAligner and PPP model. PPP^1 refers to Needleman-Wunsch gap penalty parameters set to $g = 30$, $e = 20$ and PPP^2 to $g = 20$, $e = 10$. The third column lists the number of different zinc finger array pairs aligned; fourth column lists the number of wrongly aligned motifs.

In our tests, we achieved the best results when the gap opening penalty g was set to 30 and the gap extension penalty e to 20. The results of all tests are shown in the Table 3. PPP^1 was able to outperform the MotifAligner on the *Unchanged* and *1/5 Loss* datasets. On the other hand, our model performed slightly worse as the number of lost fingers was increased.

4 Conclusion

We have designed and implemented an algorithm for alignment of sequences with repetitive motifs. The algorithm is built on top of two types of hidden Markov models. It utilizes positional information from two copies of a profile HMM and uses a pair HMM to align the motif sequences. We were able to apply our model on real world data, and obtained better results than the only existing program specifically designed to align sequences with repetitive motifs.

There is still a room for improvement of our work. Apart from obvious upgrades, like a more efficient implementation, the underlying model can be enhanced in several ways. For example, an interesting question is whether some other scoring function of individual motif alignments would perform better. Such a function might be based on different properties of the underlying models, e.g. the full probability of a sequence, instead of the probability of the Viterbi path.

To alleviate problems caused by the computational complexity of the algorithm, various heuristics could be applied, especially methods avoiding exhausting computations of the whole dynamic programming matrix. In order to apply our model to other protein families with repeating motifs, a more robust procedure for parameter estimation should be established. In addition, a method for assessment of statistical significance of alignments may be helpful when computing alignments of large datasets where random similarities are more likely to occur.

The model we have implemented is not the only way of doing sequence alignment with repetitive motifs. It is very appealing to use a monolithic probabilistic model instead of multiplying probabilities of three separate models. We have tried to develop such a model, but we were not able to overcome some of their intrinsic difficulties.

From the practical point of view, the most serious problem we have encountered is the lack of reliable benchmark for assessing the accuracy of alignments with repetitive motifs. A high quality reference is very valuable, because it allows exact evaluation of algorithms and can give a clue where are the weak and the strong parts of a particular method, or how to set the method parameters to ensure optimal performance. We hope that our work will at least partially serve as a catalyst towards the creation of such a resource.

References

1. A. Bateman, S. Boehm, E. L. L. Sonnhammer, F. Gago: *Mulitple sequence alignment of zinc finger C2H2 type family.* Pfam Family: zf-C2H2 (PF00096), 2011. Online. http://pfam.sanger.ac.uk/family/PF00096.
2. E. J. Bellefroid, D. A. Poncelet, P. J. Lecocq, O. Revelant, J. A. Martial: *The evolutionarily conserved Krppel-associated box domain defines a subfamily of eukaryotic multifingered proteins.* Proc. Natl. Acad. Sci. U.S.A. 88(9), 1991, 3608–3612.
3. G. Ding, P. Lorenz, M. Kreutzer, Y. Li, H. J. Thiesen: *SysZNF: the C2H2 zinc finger gene database.* Nucleic Acids Res. 37 (Database issue), 2009, D267–273.

4. R. Durbin, S. Eddy, A. Krogh, G. Mitchison: *Biological sequence analysis.* 1st edition. Cambridge University Press. 1998, 356 p., ISBN: 978-0521629713.
5. S. R. Eddy: *Accelerated profile HMM searches.* PLoS Comput. Biol. 7(10), 2011, e1002195.
6. I. Elias: *Settling the intractability of multiple alignment.* Journal of Computational Biology 13(7), 2006, 1323–1339.
7. A. T. Hamilton, S. Huntley, M. Tran-Gyamfi, D. M. Baggott, L. Gordon, L. Stubbs: *Evolutionary expansion and divergence in the ZNF91 subfamily of primate-specific zinc finger genes.* Genome Res. 16(5), 2006, 584–594.
8. S. Henikoff, J. G. Henikoff: *Amino acid substitution matrices from protein blocks.* Proc. Natl. Acad. Sci. U.S.A., 89(22), 1992, 10915–10919.
9. S. Huntley, D. M. Baggott, A. T. Hamilton, M. Tran-Gyamfi, S. Yang, J. Kim, L. Gordon, E. Branscomb, L. Stubbs: *A comprehensive catalog of human KRAB-associated zinc finger genes: insights into the evolutionary history of a large family of transcriptional repressors.* Genome Res., 16(5), 2006, 669–677.
10. S. B. Needleman, C. D. Wunsch: *A general method applicable to the search for similarities in the amino acid sequence of two proteins.* J. Mol. Biol., 48(3), 1970, 443–453.
11. K. Nowick, C. Fields, T. Gernat, D. Caetano-Anolles, N. Kholina, L. Stubbs: *Gain, loss and divergence in primate zinc-finger genes: a rich resource for evolution of gene regulatory differences between species.* PLoS ONE 6(6), 2011, e21553.
12. K. Nowick, A. T. Hamilton, H. Zhang, L. Stubbs: *Rapid sequence and expression divergence suggest selection for novel function in primate-specific KRAB-ZNF genes.* Molecular Biology and Evolution 27(11), 2010, 2606–2617.
13. P. Rice, I. Longden, A. Bleasby: *EMBOSS: the European molecular biology open software suite.* Trends Genet., 16(6), 2000, 276–277.
14. D. Schmidt, R. Durrett: *Adaptive evolution drives the diversification of zinc-finger binding domains.* Mol. Biol. Evol. 21(12), 2004, 2326–2339.
15. B. Schuster-Bockler, J. Schultz, S. Rahmann: *HMM Logos for visualization of protein families.* BMC Bioinformatics 5, 2004, 7.
16. J. H. Thomas, R. O. Emerson: *Evolution of C2H2-zinc finger genes revisited.* BMC Evol. Biol. 9, 2009, 51.
17. UCSC Human Genome Feb. 2009 (hg19, GRCh37) Pairwise Alignments. Online. http://hgdownload.cse.ucsc.edu/downloads.html.
18. R. Urrutia: *KRAB-containing zinc-finger repressor proteins.* Genome Biology 4(10), 2003, 231.
19. A. J. Viterbi: *Error bounds for convolutional codes and an asymtotically optimum decoding algorithm.* IEEE Transactions on Information Theory IT-13, 1967, 260–267.

Surrogate solutions of Fredholm equations by feedforward networks[⋆]

Věra Kůrková

Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague
vera@cs.cas.cz, http://www.cs.cas.cz/∼vera

Abstract. *Surrogate solutions of Fredholm integral equations by feedforward neural networks are investigated theoretically. Convergence of surrogate solutions computable by networks with increasing numbers of computational units to theoretically optimal solutions is proven and upper bounds on rates of convergence are derived. The results hold for a variety of computational units, they are illustrated by examples of perceptrons and Gaussian radial units.*

1 Introduction

Surrogate modeling is one of successful applications of neural networks. Often it has been used for empirical functions, i.e., functions for which no mathematical formulas are known and thus their values can only be gained experimentally. When such experimental evaluations are too expensive or time consuming, it can be useful to perform them merely for some samples of points of the domains of the empirical functions and the obtained values use as training data for neural networks. The networks trained on such data can play roles of *surrogate models* of these empirical functions. For example, input-output functions of feedforward networks have been used in chemistry as surrogate models of empirical functions assigning to compositions of chemicals measures of quality of catalyzers produced by reactions of these chemicals, in biology as models of empirical functions classifying structures of RNA, and in economy as models of functions assigning credit ratings to companies [7, 2]. However, it should be emphasized that results obtained by surrogate modeling of empirical functions can only be used as suggestions to be confirmed by additional experiments as no other than empirical knowledge of the functions is available. Moreover, no methodology for choice of suitable network architectures, type of computational units and their number has been developed.

In contrast to the case of empirical functions, analytically described functions, which are subjects of surrogate modeling due to their complicated and time consuming numerical calculations, provide a potential for theoretical analysis of quality of their surrogate models. Available analytic expressions can be compared with various surrogate models. One can investigate mathematical properties of these functions as well as properties of their surrogate models aiming to estimate speed of convergence of approximations computable by surrogate models with increasing model complexity to functions described by the complicated formulas. Mathematical theory of approximation of functions by neural networks offers some tools for derivation of such estimates.

A large class of functions described by mathematical formulas, numerical calculations of which are difficult, is formed by solutions of Fredholm integral equations. These equations play an important role in many problems in applied science and engineering. They arise in image restoration, differential problems with auxiliary boundary conditions, potential theory and elasticity, etc. (see, e.g., [23, 22, 24]). Mathematical descriptions of solutions of Fredholm equations following from classical Fredholm theorem [27, p.499] involve complicated expressions in terms of infinite Liouville-Neumann series with coefficients in the forms of integrals. Thus numerical calculations of these expressions are time consuming.

Recently, several authors [13, 6] explored experimentally possibilities of surrogate modeling of solutions of Fredholm equations by perceptron and kernel networks. Motivated by these experimental studies, in [9] we initiated a theoretical analysis of approximation of solutions of Fredholm equtions by neural networks. In [9, 12, 20], estimates of rates of approximation were derived for surrogate modeling by networks with kernel units induced by the same kernels as the kernels defining the equations and extended to certain smooth kernels.

In this paper, we investigate surrogate solutions of Fredholm integral equations by networks with general computational units. Taking advantage of results from nonlinear approximation theory and suitable integral representations of functions in the form of "infinite" networks, we estimate how well surrogate solutions computable by feedforward networks can approximate exact solutions of Fredholm equations. We apply general results to perceptron and Gaussian radial networks.

[⋆] This work was partially supported by MŠMT grant COST INTELLI OC10047 and RVO 67985807.

The paper is organized as follows. In section 2, we describe surrogate modeling of functions by feedforward neural networks and in section 3, we introduce Fredholm integral equations and theoretical approach to their solutions. In section 4, we recall some results from nonlinear approximation theory and apply them to approximation of solutions of Fredholm equations by feedorward networks. We illustrate our results by an example of approximation of Fredholm equations with the Gaussian kernel by networks with perceptrons and Gaussian radial units.

2 Surrogate modeling by neural networks

A traditional approach to surrogate modeling of functions has employed linear methods such as polynomial interpolation. For suitable points $x_1, \dots, x_m$ from a domain $X \subset \mathbb{R}^d$, empirically or numerically obtained approximations $\bar{\phi}(x_1), \dots, \bar{\phi}(x_m)$ of values $\phi(x_1), \dots, \phi(x_m)$ of a function ϕ are interpolated by functions from n-dimensional function spaces. These spaces are obtained as *linear spans*

$$\operatorname{span}\{g_1, \dots, g_n\} := \left\{ \sum_{i=1}^{n} w_i g_i \,|\, w_i \in \mathbb{R} \right\}, \qquad (1)$$

where the functions $g_1, \dots g_n$ are first n elements from a set $G = \{g_n \,|\, n \in \mathbb{N}_+\}$ with a *fixed linear ordering* (we use the standard notation := meaning a definition). Typical examples of linear approximators are algebraic or trigonometric polynomials. They are obtained by linear combinations of powers of increasing degrees or trigonometric functions with increasing frequencies, resp.

Feedforward neural networks have more adjustable parameters than linear models as in addition to coefficients of linear combinations of basis functions, also inner coefficients of computational units are optimized during learning. Thus they are sometimes called *variable-basis approximation schemas* in contrast to traditional linear approximators which are called *fixed basis approximation schemas.* In some cases, especially in approximation of functions of large numbers of variables, it was proven that neural networks achieve better approximation rates than linear models with much smaller model complexity [11, 10].

One-hidden-layer networks with one linear output unit compute input-output functions from sets of the form

$$\operatorname{span}_n G := \left\{ \sum_{i=1}^{n} w_i g_i \,|\, w_i \in \mathbb{R}, g_i \in G \right\}, \qquad (2)$$

where the set G is sometimes called a *dictionary* [14] and n is the *number of hidden computational units.* This number can be interpreted as a measure of *model complexity* of the network. In contrast to linear approximation, the dictionary G has no fixed ordering.

Often, dictionaries are parameterized families of functions modeling computational units, i.e., they are of the form

$$G_F(X, Y) := \{F(\cdot, y) : X \to \mathbb{R} \,|\, y \in Y\}\ , \qquad (3)$$

where $F{:}X \times Y \to \mathbb{R}$ is a function of two variables, an input vector $x \in X \subseteq \mathbb{R}^d$ and a parameter $y \in Y \subseteq \mathbb{R}^s$. When $X = Y$, we write briefly $G_F(X)$. So one-hidden-layer networks with n units from a dictionary $G_F(X,Y)$ compute functions from the set

$$\operatorname{span}_n G_F(X, Y) := \left\{ \sum_{i=1}^{n} w_i F(x, y_i) \,|\, w_i \in \mathbb{R}, y_i \in Y \right\}.$$

In some contexts, F is called a *kernel.* However, the above-described computational scheme includes fairly general computational models, such as functions computable by perceptrons, radial or kernel units, Hermite functions, trigonometric polynomials, and splines. For example, with

$$F(x, y) = F(x, (v, b)) := \sigma(\langle v, x\rangle + b)$$

and $\sigma : \mathbb{R} \to \mathbb{R}$ a sigmoidal function, the computational scheme (2) describes one-hidden-layer *perceptron networks. Radial (RBF) units* with an activation function $\beta : \mathbb{R} \to \mathbb{R}$ are modeled by the kernel

$$F(x, y) = F(x, (v, b)) := \beta(v\|x - b\|).$$

Typical choice of β is the Gaussian function. *Kernel units* used in support vector machine (SVM) have the form $F(x, y)$ where $F : X \times X \to \mathbb{R}$ is a symmetric positive semidefinite function [27].

Various learning algorithms optimize parameters $y_1, \dots, y_n$ of computational units as well as coefficients $w_1, \dots, w_n$ of their linear combinations so that network input-output functions

$$\sum_{i=1}^{n} w_i\, F(., y_i)$$

from the set $\operatorname{span}_n G_F(X, Y)$ fit well to training samples $\{(x_i, \bar{\phi}(x_i) \,|i = 1, \dots, m\}$.

3 Solutions of Fredholm integral equations

Solving an *inhomogeneous Fredholm integral equation of the second kind* on a domain $X \subseteq \mathbb{R}^d$ for a given $\lambda \in \mathbb{R} \setminus \{0\}$, $K : X \times X \to \mathbb{R}$, and $f : X \to \mathbb{R}$ is a task of finding a function $\phi : X \to \mathbb{R}$ such that for all $x \in X$

$$\phi(x) - \lambda \int_X \phi(y) K(x,y)\, dy = f(x). \tag{4}$$

The function ϕ is called *solution*, f *data*, K *kernel*, and λ *parameter* of the equation (4).

Fredholm equations can be described in terms of theory of inverse problems. Formally, an *inverse problem* is defined by a linear operator $A : \mathcal{X} \to \mathcal{Y}$ between two function spaces. It is a task of finding for $f \in \mathcal{Y}$ (called *data*) some $\phi \in \mathcal{X}$ (called *solution*) such that

$$A(\phi) = f.$$

Let T_K denotes the integral operator with a kernel $K : X \times X \to \mathbb{R}$ defined for every ϕ in a suitable function space as

$$T_K(\phi)(x) := \int_X \phi(y)\, K(x,y)\, dy \tag{5}$$

and $I_{\mathcal{X}}$ denotes the identity operator. Then the Fredholm equation (4) can be represented as an inverse problem defined by the linear operator $I_{\mathcal{X}} - \lambda T_K$. So it is a problem of finding for a given data f a solution ϕ such that

$$(I_{\mathcal{X}} - \lambda T_K)(\phi) = f. \tag{6}$$

The classical Fredholm alternative theorem from 1903 proved existence and uniqueness of solutions of Fredholm equations for continuous one variable functions on intervals. A modern version holding for general Banach spaces is stated in the next theorem from [27, p.499]. Recall that an operator $T : (\mathcal{X}, \|.\|_{\mathcal{X}}) \to (\mathcal{Y}, \|.\|_{\mathcal{Y}})$ between two Banach spaces is called *compact* if it maps bounded sets to precompact sets (i.e., sets whose closures are compact).

Theorem 1. *Let $(\mathcal{X}, \|.\|_{\mathcal{X}})$ be a Banach space, $T : (\mathcal{X}, \|.\|_{\mathcal{X}}) \to (\mathcal{X}, \|.\|_{\mathcal{X}})$ be a compact operator, and $I_{\mathcal{X}}$ be the identity operator. Then the operator $I_{\mathcal{X}} + T : (\mathcal{X}, \|.\|_{\mathcal{X}}) \to (\mathcal{X}, \|.\|_{\mathcal{X}})$ is one-to-one if and only if it is onto.*

A straightforward corollary of Theorem 1 guarantees existence and uniqueness of solutions of the inverse problem (6) when T is a compact operator and $1/\lambda$ is not its eigenvalue (i.e., there is no $\phi \in \mathcal{X}$ for which $T(\phi) = \frac{\phi}{\lambda}$).

Corollary 1. *Let $(\mathcal{X}, \|.\|_{\mathcal{X}})$ be a Banach space, $T : (\mathcal{X}, \|.\|_{\mathcal{X}}) \to (\mathcal{X}, \|.\|_{\mathcal{X}})$ be a compact operator, $I_{\mathcal{X}}$ be the identity operator, and $\lambda \neq 0$ be such that $1/\lambda$ is not an eigenvalue of T. Then the operator $I_{\mathcal{X}} - \lambda T$ is invertible (one-to-one and onto).*

The following proposition gives conditions guaranteeing compactness of operators T_K in spaces $(\mathcal{C}(X), \|.\|_{\sup})$, where $X \subseteq \mathbb{R}^d$, of bounded continuous functions on X with the supremum norm $\|f\|_{\sup} = \sup_{x \in X} |f(x)|$ and to spaces $(\mathcal{L}^2(X), \|.\|_{\mathcal{L}^2})$ of square integrable functions with the norm $\|f\|_{\mathcal{L}^2} = \left(\int_X f(x)^2\, dx\right)^{1/2}$. The proof is well-known and easy to check (see, e.g., [26, p. 112]).

Proposition 1. *(i) If $X \subset \mathbb{R}^d$ is compact and $K : X \times X \to \mathbb{R}$ is continuous, then $T_K : (\mathcal{C}(X), \|.\|_{\sup}) \to (\mathcal{C}(X), \|.\|_{\sup})$ is a compact operator.*
(ii) If $X \subset \mathbb{R}^d$ and $K \in \mathcal{L}^2(X \times X)$, then $T_K : (\mathcal{L}^2(X), \|.\|_{\mathcal{L}^2}) \to (\mathcal{L}^2(X), \|.\|_{\mathcal{L}^2})$ is a compact operator.

So by Corollary 1, when the assumptions of the Proposition 1 (i) or (ii) are satisfied and $1/\lambda$ is not an eigenvalue of T_K, then for every f in $\mathcal{C}(X)$ or $\mathcal{L}^2(X)$, resp., there exists unique solution ϕ of the equation (4). It is known (see, e.g, [1]) that the solution ϕ can be expressed as

$$\phi(x) = f(x) - \lambda \int_X f(y)\, R_K^\lambda(x,y)\, dy\,, \tag{7}$$

where $R_K^\lambda : X \times X \to \mathbb{R}$ is called a *resolvent kernel* . However, the formula expressing the resolvent kernel is not suitable for efficient computation as it is expressed as an infinite Neumann series in powers of λ with coefficients in the form of integrals with iterated kernels [5, p.140]. So numerical calculations of values of solutions of Fredholm equations based on (7) are quite computationally demanding. Thus various methods of finding surrogate solutions of (4) have been explored [13, 6]. Some of these methods utilized feedforward networks. Such networks were trained on samples of input-output pairs $\{(x_1, \bar\phi(x_1)), \ldots, (x_m, \bar\phi(x_m)\}$ where $\{x_1, \ldots, x_m\}$ are selected points from the domain X and $\{\bar\phi(x_1), \ldots, \bar\phi(x_m)\}$ are numerically computed approximations of values $\{\phi(x_1), \ldots, \phi(x_m)\}$ of the solution ϕ. In these experiments, one-hidden-layer networks with perceptrons and Gaussian radial units were used. However, without a theoretical analysis, it is not clear how to choose a proper number n of network units to guarantee that input-output functions approximate well the solution and the networks are not too large to make their implementation unfeasible.

4 Rates of convergence of surrogate solutions

Estimates of model complexity of one-hidden-layer networks approximating solutions of Fredholm equations follow from inspection of upper bounds on rates of decrease of errors in approximation of solutions of the equation (4) by sets $\mathrm{span}_n G$ with n increasing. Approximation properties of sets of the form $\mathrm{span}_n\, G$ have been studied in mathematical theory of neurocomputing for various types of dictionaries G and norms measuring approximation errors such as Hilbert-space norms and the supremum norm (see, e.g., [4, 8]). Some such bounds have the form $\frac{\xi(h)}{\sqrt{n}}$, where n is the number of network units and $\xi(h)$ depends on a certain norm of the function h to be approximated.

This norm is tailored to the dictionary of computation units and can be estimated for functions satisfying suitable integral equations. The norm is defined quite generally for any bounded nonempty subset G of a normed linear space $(\mathcal{X}, \|.\|_{\mathcal{X}})$. It is called *G-variation*, denoted $\|.\|_G$, and defined for all $f \in \mathcal{X}$ as

$$\|f\|_{G,\mathcal{X}} := \inf \left\{c > 0 \,|\, f/c \in \mathrm{cl}_{\mathcal{X}}\, \mathrm{conv}\, (G \cup -G)\right\},$$

where the closure $\mathrm{cl}_{\mathcal{X}}$ is taken with respect to the topology generated by the norm $\|.\|_{\mathcal{X}}$ and conv denotes the *convex hull*. So G-variation depends on the ambient space norm, but when it is clear from the context, we write merely $\|f\|_G$ instead of $\|f\|_{G,\mathcal{X}}$.

The concept of variational norm was introduced by Barron [3] for sets of characteristic functions. Among them, the set of characteristic functions of half-spaces forming the dictionary of functions computable by Heaviside perceptrons. Barron's concept was generalized in [18, 19] to variation with respect to an arbitrary bounded set of functions and applied to various dictionaries of computational units such as Gaussian RBF units or kernel units [16].

The following theorem on rates of approximation by sets of the form $\mathrm{span}_n G$ is a reformulation from [19] of results by Maurey [25], Jones [15], Barron [4] in terms of G-variation. For a normed space $(\mathcal{X}, \|.\|_{\mathcal{X}})$, $g \in \mathcal{X}$ and $A \subset \mathcal{X}$, we denote by

$$\|g - A\|_{\mathcal{X}} := \inf_{f \in A} \|g - f\|_{\mathcal{X}}$$

the *distance* of g from A.

Theorem 2. *Let $(\mathcal{X}, \|.\|_{\mathcal{X}})$ be a Hilbert space, G its bounded nonempty subset, $s_G = \sup_{g\in G}\|g\|_{\mathcal{X}}$, $f \in \mathcal{X}$, and n be a positive integer. Then*

$$\|h - \mathrm{span}_n G\|_{\mathcal{X}}^2 \leq \frac{s_G^2\|h\|_G^2 - \|h\|_{\mathcal{X}}^2}{n}.$$

Theorem 2 guarantees that for every $\varepsilon > 0$ and n satisfying

$$n \geq \left(\frac{s_G\, \|h\|_G}{\varepsilon}\right)^2,$$

a network with n units computing functions from the dictionary G approximates the function h within ε. So the size of G-variation of the function h to be approximated is a critical factor influencing model complexity of networks approximating h within a required accuracy. Generally, it is not easy to estimate G-variation. However, the following theorem from [21] shows that for the special case of functions with integral representations in the form of "infinite networks", variational norms are bounded from above by the $\mathcal{L}^1$-norms of "output-weight" functions of these networks.

Theorem 3. *Let $X \subseteq \mathbb{R}^d$, $Y \subseteq \mathbb{R}^s$, $w \in \mathcal{L}^1(Y)$, $K : X \times Y \to \mathbb{R}$ be such that $G_K(X,Y) = \{K(.,y) \,|\, y \in Y\}$ is a bounded subset of $(\mathcal{L}^2(X), \|.\|_{\mathcal{L}^2})$, and $h \in \mathcal{L}^2(X)$ be such that for all $x \in X$, $h(x) = \int_Y w(y)\, K(x,y)\, dy$. Then*

$$\|h\|_{G_K(X,Y)} \leq \|w\|_{\mathcal{L}^1}.$$

To apply Theorem 2 to approximation of solutions of Fredholm equations by surrogate models formed by networks with units from a general dictionary G, we need upper bounds on G-variation. The next proposition describes a relationship between variations with respect to two sets, G and F; its proof follows easily from the definition of variational norm.

Proposition 2. *Let $(\mathcal{X}, \|.\|_{\mathcal{X}})$ be a normed linear space, F and G its bounded subsets such that $c_{G,F} := \sup_{g\in G}\|g\|_F < \infty$. Then for all $h \in \mathcal{X}$, $\|h\|_G \leq c_{G,F}\, \|h\|_F$.*

Combining Theorems 2, 3, and Proposition 2, we obtain the next theorem on rates of approximation of functions which can be expressed as $h = T_K(w)$ by networks with units from a dictionary G.

Theorem 4. *Let $X \subseteq \mathbb{R}^d$, $K : X \times Y \to \mathbb{R}$ be a bounded kernel, and $h \in \mathcal{L}^2(X)$ such that $h = T_K(w) = \int_Y w(y)K(.,y)\, dy$ for some $w \in \mathcal{L}^1(Y)$, where $G_K(X,Y)$ is a bounded subset of $\mathcal{L}^2(X)$. Let G be a bounded subset of $\mathcal{L}^2(X)$ with $s_G = \sup_{g\in G}\|g\|_{\mathcal{L}^2}$ such that $c_{G,K} = \sup_{g\in G}\|g\|_{G_K(X,Y)}$ is finite. Then for all $n > 0$,*

$$\|h - \mathrm{span}_n\, G\|_{\mathcal{L}^2} \leq \frac{s_G\, c_{G,K}\, \|w\|_{\mathcal{L}^1}}{\sqrt{n}}.$$

A critical factor in the estimate given in Theorem 4 is the $\mathcal{L}^1$-norm of the "output-weight function" w in the representation of the function h to be approximated an "infinite network" with units computing $K(.,y)$ in the form

$$h(x) = T_K(w) = \int_Y w(y)\, K(x,y)\, dy.$$

The solution ϕ of the Fredholm equation minus the function f representing the data, $\phi - f$, is the image of $\lambda\,\phi$ mapped by the integral operator T_K, i.e.,

$$\phi - f = T_K(\lambda\,\phi) = \lambda \int_X \phi(y) K(x,y)\, dy\,.$$

Thus to apply Theorem 4 to approximation of a solution of Fredholm equation, we need to estimate the $\mathcal{L}^1$-norm of the solution ϕ itself as $\lambda\,\phi$ plays the role of the "output-weight" function in the infinite network $\int_X \lambda\,\phi(y)\,K(x,y)\,dy$.

Theorem 5. *Let $X \subset \mathbb{R}^d$ be compact, $K : X \times X \to \mathbb{R}$ be a bounded kernel such that $K \in \mathcal{L}^2(X \times X)$, $\rho_K := \int_X \sup_{y\in X}|K(x,y)|dx$ be finite, G be a bounded subset of $\mathcal{L}^2(X)$ with $s_G = \sup_{g\in G}\|g\|_{\mathcal{L}^2}$ such that $c_{G,K} = \sup_{g\in G}\|g\|_{G_K(X)}$ is finite, and $\lambda \neq 0$ be such that $\frac{1}{\lambda}$ is not an eigenvalue of T_K and $|\lambda|\,\rho_K < 1$. Then the solution ϕ of the equation (4) satisfies for all $n > 0$,*

$$\|\phi - f - \mathrm{span}_n\, G\|_{\mathcal{L}^2} \leq \frac{s_G\, c_{G,K}\, |\lambda|\, \|f\|_{\mathcal{L}^1}}{(1 - |\lambda|\,\rho_K)\,\sqrt{n}}\,.$$

Proof. As $\phi - f$ satisfies the Fredholm equation (4), we have for every $x \in X$,

$$|\phi(x)| \leq |\lambda|\,\|\phi\|_{\mathcal{L}^1}\sup_{y\in X}|K(x,y)| + |f(x)|.$$

Integrating over X we get

$$\|\phi\|_{\mathcal{L}^1} \leq |\lambda|\,\rho_K\,\|\phi\|_{\mathcal{L}^1} + \|f\|_{\mathcal{L}^1}$$

and so $\|\phi\|_{\mathcal{L}^1}\,(1 - |\lambda|\,\rho_K) \leq \|f\|_{\mathcal{L}^1}$. This inequality is non trivial only when $|\lambda| < \frac{1}{\rho_K}$. Thus we get $\|w\|_{\mathcal{L}^1} = |\lambda|\|\phi\|_{\mathcal{L}^1} \leq \frac{|\lambda|\,\|f\|_{\mathcal{L}^1}}{1-|\lambda|\,\rho_K}$. The statement then follows from Theorem 4. □

Theorem 5 estimates rates of approximation of the function $\phi - f = \lambda \int_X f(y)\, R_K^\lambda(x,y)\, dy$ by functions computable by networks with units from dictionary G formed by functions with G_K-variations bounded by $c_{G,K}$. Numerical computations of values of the function $\lambda \int_X f(y)\, R_K^\lambda(x,y)\, dy$ are time consuming. For $|\lambda| < \frac{1}{\rho_K}$ and any bounded dictionary G with finite bound $c_{G,K}$ on $G_{K(X)}$-variations on its elements, input-output functions of networks with increasing numbers of units from G converge to the function $\phi - f$. When for a reasonable size of the network measured by the number n of units, the upper bound from Theorem 5 is sufficiently small, the network can serve as a good surrogate model of the solution of Fredholm equation.

To illustrate our results, consider approximation of Fredholm equations with the Gaussian kernel

$$K_b(x,y) = e^{-b\|x-y\|}$$

with the width b by surrogate solutions in the form of input-output functions of networks with two types of popular units: sigmoidal perceptrons and Gaussian radial units. Note that Fredholm equations with Gaussian kernels arise, e.g., in image restoration problems [24]. By μ is denoted the *Lebesgue measure* on $\mathbb{R}^d$ and by $P_d^\sigma(X)$ the *dictionary of functions on X computable by sigmoidal perceptrons.*

Corollary 2. *Let $X \subset \mathbb{R}^d$ be compact, $b > 0$, $K_b(x,y) = e^{-b\|x-y\|^2}$, $\lambda \neq 0$ be such that $\frac{1}{\lambda}$ is not an eigenvalue of T_{K_b} and $|\lambda| < 1$. Then the solution ϕ of the equation (4) with f continuous satisfies for all $n > 0$*

$$\|\phi - f - \mathrm{span}_n\, G_{K_b}(X)\|_{\mathcal{L}^2} \leq \frac{\mu(X)\,|\lambda|\,\|f\|_{\mathcal{L}^1}}{(1 - |\lambda|\,\mu(X)\,)\,\sqrt{n}}$$

and

$$\|\phi - f - \mathrm{span}_n\, P_d^\sigma(X)\|_{\mathcal{L}^2} \leq \frac{\mu(X)\,2d\,|\lambda|\,\|f\|_{\mathcal{L}^1}}{(1 - |\lambda|\,\mu(X)\,)\,\sqrt{n}}\,.$$

Proof. It was shown in [17] that variation of the d-dimensional Gaussian with respect to the dictionary formed by sigmoidal perceptrons is bounded from above by $2d$ and thus by Proposition 2, $c_{P_d^\sigma,K_b} \leq 2d$. The statement then follows by Theorem 5, an estimate $s_{G_{K_b}} \leq \mu(X)$ and equalities $s_{P_d^\sigma} = \mu(X)$ and $\rho_{K_b} = \mu(X)$. □

References

1. K. Atkinson: *The numerical solution of integral equations of the second kind.* Cambridge University Press, 1997.
2. M. Baerns, M. Holeňa: *combinatorial development of solid catalytic materials.* Imperial College Press, London, 2009.
3. A. R. Barron: *Neural net approximation.* In K. Narendra, (Ed.), Proc. 7th Yale Workshop on Adaptive and Learning Systems, Yale University Press, 1992.
4. A. R. Barron: *Universal approximation bounds for superpositions of a sigmoidal function.* IEEE Transactions on Information Theory 39, 1993, 930–945.
5. R. Courant, D. Hilbert: *Methods of mathematical physic*, volume I. Wiley, New York, 1989.
6. S. Effati, R. Buzhabadi: *A neural network approach for solving Fredholm integral equations of the second kind.* Neural Computing and Applications. doi:10.1007/s00521-010-0489-y.
7. A. Forrester, A. Sobester, A. Keane: *Engineering design via surrogate modelling: A practical guide.* Wiley, 2008.
8. F. Girosi: *Approximation error bounds that use VC-bounds.* In Proceedings of ICANN 1995, Paris, 1995, 295–302.

9. G. Gnecco, V. Kůrková, M. Sanguineti: *Bounds for approximate solutions of Fredholm integral equations using kernel networks.* In T. Honkela and et al., (Eds), Lecture Notes in Computer Science (Proceedings of ICANN 2011), vol. 6791, Springer, Heidelberg, 2011, 126–133.
10. G. Gnecco, V. Kůrková, M. Sanguineti: *Can dictionary-based computational models outperform the best linear ones?* Neural Networks 24, 2011, 881–887.
11. G. Gnecco, V. Kůrková, M. Sanguineti: *Some comparisons of complexity in dictionary-based and linear computational models.* Neural Networks, 24, 2011, 171–182.
12. G. Gnecco, V. Kůrková, M. Sanguineti: *Accuracy of approximations of solutions to Fredholm equations by kernel methods.* Applied Mathematics and Computation 218, 2012, 7481–7497.
13. A. Golbabai, S. Seifollahi: *Numerical solution of the second kind integral equations using radial basis function networks.* Applied Mathematics and Computation 174, 2006, 877–883.
14. R. Gribonval, P. Vandergheynst: *On the exponential convergence of matching pursuits in quasi-incoherent dictionaries.* IEEE Transactions on Information Theory 52, 2006, 255–261.
15. L. K. Jones: *A simple lemma on greedy approximation in Hilbert space and convergence rates for projection pursuit regression and neural network training. Annals of Statistics*, 20:608–613, 1992.
16. P. C. Kainen, V. Kůrková, M. Sanguineti: *Complexity of Gaussian radial-basis networks approximating smooth functions.* Journal of Complexity 25, 2009, 63–74.
17. P. C. Kainen, V. Kůrková, A. Vogt: *A Sobolev-type upper bound for rates of approximation by linear combinations of Heaviside plane waves.* Journal of Approximation Theory 147, 2007, 1–10.
18. V. Kůrková: *Dimension-independent rates of approximation by neural networks.* In K. Warwick and M. Kárný, (Eds), Computer-Intensive Methods in Control and Signal Processing. The Curse of Dimensionality, Birkhäuser, Boston, MA, 1997, 261–270.
19. V. Kůrková: *High-dimensional approximation and optimization by neural networks.* In J. Suykens, G. Horváth, S. Basu, C. Micchelli, J. Vandewalle, (Eds), Advances in Learning Theory: Methods, Models and Applications, (Chapter 4). IOS Press, Amsterdam, 2003, 69–88.
20. V. Kůrková: *Accuracy estimates for surrogate solutions of integral equations by neural networks.* In M. Bieliková, G. Fridrich, G. Gottlob, S. Katzenbeisser, R. Špánek, G. Turán, (Eds), SOFSEM 2012: Theory and Practice of Computer Science, vol. II,. Institute of Computer Science, Prague, 2012 , 95–102.
21. V. Kůrková: *Complexity estimates based on integral transforms induced by computational units.* Neural Networks, 30:160–167, 2012.
22. A. T. Lonseth: *Sources and applications of integral equations.* SIAM Review 19, 1977, 241–278.
23. W. V. Lovitt: *Linear integral equations.* Dover, New York, 1950.
24. Y. Lu, L. Shen, Y. Xu: *Integral equation models for image restoration: high accuracy methods and fast algorithms.* Inverse Problems, 26. doi: 10.1088/0266-5611/26/4/045006.
25. G. Pisier: *Remarques sur un résultat non publié de B. Maurey.* In Séminaire d'Analyse Fonctionnelle 1980-81, vol. I, no. 12, École Polytechnique, Centre de Mathématiques, Palaiseau, France, 1981.
26. W. Rudin: *Functional analysis.* McGraw-Hill, Boston, 1991.
27. I. Steinwart, A. Christmann: *Support vector machines.* Springer, New York, 2008.

Localization of (in)consistencies by monotone reducing automata*

Martin Procházka and Martin Plátek

Charles University, MFF UK, Department of Computer Science
Malostranské náměstí 25, 118 00 Praha 1, Czech Republic prma@centrum.cz, martin.platek@mff.cuni.cz

Abstract. *A reducing automaton (red-automaton) is a deterministic automaton proposed for checking word and sub-word correctness by the use of analysis by reduction. Its monotone version characterizes the class of deterministic context-free languages (DCFL). We propose a method for a construction of a deterministic monotone enhancement of any monotone reducing automaton which is able with the help of special auxiliary symbols to localize its prefix and post-prefix (in)consistencies, and certain types of reducing conflicts. In other words this method ensures a robust analysis by reduction without spurious error messages. We formulate natural conditions for which this method ensures the localization of all prefix and post-prefix inconsistencies in any (incorrect) word with respect to a DCFL.*

1 Introduction

A reducing automaton (red-automaton for short) is a device that models the so called analysis by reduction. Analysis by reduction consists in a stepwise simplification of an extended sentence (word) until a simple sentence (word) is obtained or an error is found. It is based on another automata model – restarting automaton (R-automaton) introduced in [2]. Similarly to R-automaton, red-automaton can only delete symbols. At some place it decides to delete some of the last k visited symbols, where k is limited by a fixed constant and then restarts its computations, i.e. it enters its initial state and its head is placed on the left end of the remaining word.

Reducing automaton is formalized as an extension of deterministic finite automaton. This kind of formalization serves here as a basic tool for the method of algorithmic localization of syntactic inconsistencies (errors) for the languages from the class of DCFL. The notion of red-automata was introduced in order to present naturaly the techniques of minimization. In [7] we construct to any red-automaton M an unambiguously determined minimal red-automaton M_m which preserves the recognized language, and the set of all reductions defined by M.

For a given language L and a word $w \notin L$, it is natural to define the maximal correct prefix and prefix-inconsistency (prefix error) in w. The prefix inconsistency is the minimal incorrect prefix of w. Let $x \in \Sigma$ be the leftmost symbol in the word w such that $w = uxv$, $u, v \in \Sigma^*$, there exists a word $v' \in \Sigma^*$ for which it holds $uv' \in L$ and there is no word v'' such that $uxv'' \in L$. The u is the maximal correct prefix of w, and ux the prefix-inconsistency of w.

In a similar way we can consider (in)correct infixes for a given language L and a word $w \notin L$. We can easily see that a prefix-inconsistence can occur in a word at most once. Our effort is to study properties of reducing automata which will ensure the detection of (in)correct prefixes, and/or certain types of (in)correct infixes. The types of the (in)correct infixes studied here are studied by different techniques already in [1]. We call them here post-prefix (in)consistencies.

In this paper we use the advantage of the fact, that to any deterministic context-free language L there is a monotone reducing automaton recognizing L which is also able to detect the prefix inconsistency (error). Such type of automaton characterizes the class of DCFL. This fact is shown in [6].

The paper is structured as follows. First, in Section 2 we introduce red-automata and their basic properties. The Section 3 creates the core of this paper. Conclusion contains some remarks about connections of the presented method with the methods based on the so called head-symbols.

2 Definitions and basic properties

The *reducing automaton* has a finite *control unit* and a *working head* attached to a list with sentinels on both ends. It works in certain cycles called *stages*. At the beginning of each stage, the head points at the leftmost item behind the left sentinel, and the control unit is in a special *initial state*. In the process of the stage the automaton moves the head from the item it currently points to the next item on the right. During such a *transition* it changes the state of its control unit according to the current state and the currently scanned symbol. The stage ends as the control unit gets to any of special states called *operations*. There are three kinds of operations: `ACC`, `ERR`, and `RED`. Both `ACC` and `ERR`-operation halts the whole computation, `ACC` accepts and `ERR` rejects the word in the list. The

* This work was supported by the grant projects of the Grant Agency of the Czech Republic No. P202/10/1333 and P103/10/0783.

$\mathtt{RED}$-operation $\mathtt{RED}(n)$ determines how the list should be shortened. Its parameter n – a binary word of a limited size – specifies which item on the left of the head are to be removed from the list. Bit $\mathtt{1}$ means "remove the item from the list", bit $\mathtt{0}$ means "leave the item in the list". After all items designated for deletion are removed, the automaton resets its control unit to the initial state and places the head at the leftmost item behind the left sentinel. The string $n \in (\mathtt{10}^*)^+$ determines, which items will be deleted from the list. If the i-th symbol of n from the right is equal to $\mathtt{1}$, then the automaton deletes the i-th item to the left from the position of the head. The item scanned by the head is considered as the first one.

All final states of a reducing automaton M create a finite subset F_M of the (unbounded) set $\{\,\mathtt{ACC}, \mathtt{ERR}\,\} \cup \{\,\mathtt{RED}(n) \mid n \in (\mathtt{1}{\cdot}\mathtt{0}^*)^+\,\}$. Now we are able to introduce reducing automata in a formal way.

A *reducing automaton (red-automaton)* is a 7-tuple $M = (\Sigma_M, \text{«}, \text{»}, S_M, s_M, F_M, f_M)$, where Σ_M is a finite input alphabet, $\text{«}, \text{»} \notin \Sigma_M$ are the (left and right) sentinels, S_M is the finite set of internal states, $s_M \in S_M$ is the (re)starting state, F_M is the finite set of final states (operations), $f_M : S_M \times (\Sigma_M \cup \{\text{»}\}) \longrightarrow (S_M \cup F_M)$ is the *transition function* of M, which fulfills the following condition:

$\forall s \in S_M : f_M(s, \text{»}) \in F_M$.

We will describe the behavior of M in more details by two functions enhancing the transition function f_M:

$\delta_M : (S_M \cup F_M \cup \{\mathtt{RED}\}) \times (\Sigma_M \cup \{\text{«}, \text{»}\}) \longrightarrow (S_M \cup F_M \cup \{\mathtt{RED}\})$

$\Delta_M : (S_M \cup F^*_M) \times (\Sigma_M \cup \{\text{«}, \text{»}\}) \longrightarrow (S_M \cup F^*_M)$

$\mathtt{RED}$ is a new (helping) state which is different from all states from S_M, and the set F^*_M is defined in the following way:

$F^*_M = F_M \cup \{\mathtt{RED}(n \cdot \mathtt{0}^k) \mid \mathtt{RED}(n) \in F_M \text{ a } k \geq 1\}$

Both functions δ_M, Δ_M for all pairs created by a state $s \in S_M$ and by a symbol $a \in (\Sigma_M \cup \{\text{»}\}$ are equal to the function f_M. We define the new functions for the remaining relevant pairs in the following way:

$$\delta_M(s, \text{«}) = s_M \qquad \Delta_M(s, \text{«}) = s_M$$

and for all $a \in (\Sigma_M \cup \{\text{»}\})$,

$$\begin{aligned} \delta_M(\mathtt{ACC}, a) &= \mathtt{ACC} & \Delta_M(\mathtt{ACC}, a) &= \mathtt{ACC} \\ \delta_M(\mathtt{ERR}, a) &= \mathtt{ERR} & \Delta_M(\mathtt{ERR}, a) &= \mathtt{ERR} \\ \delta_M(\mathtt{RED}(n), a) &= \mathtt{RED} & \Delta_M(\mathtt{RED}(n), a) &= \mathtt{RED}(n{\cdot}\mathtt{0}) \\ \delta_M(\mathtt{RED}, a) &= \mathtt{RED} \end{aligned}$$

The first enhancement of δ_M:
$\delta^*_M(s, \lambda) = s, \quad \delta^*_M(s, ua) = \delta_M(\delta^*_M(s, u), a)$

The first enhancement of Δ_M:
$\Delta^*_M(s, \lambda) = s, \quad \Delta^*_M(s, ua) = \Delta_M(\Delta^*_M(s, u), a)$

We will often use the following conventions:
$\delta^*_M(s_M, w) = \delta^*_M(\text{«}w), \quad \Delta^*_M(s_M, w) = \Delta^*_M(\text{«}w)$.

We define for the both function a further important enhancement, namely for the final subsets S of the set $S_M \cup F_M \cup \{\mathtt{RED}\}$ resp. $S_M \cup F^*_M$:

$\delta^*_M(S, u) = \{\delta^*_M(s, u) \mid s \in S\}$

$\Delta^*_M(S, u) = \{\Delta^*_M(s, u) \mid s \in S\}$

Let us note that the tuple

$(\Sigma_M \cup \{\text{«}, \text{»}\}, S_M \cup F_M \cup \{\mathtt{RED}\}, s_M, F_M, \delta_M)$ is a finite automaton which accepts exactly all prefixes (words) which lead M to some reduction, or to an acceptation, or to a rejection.

We will consider in the following only the reducing automata which fulfills the following natural condition:

$\delta^*_M(s_M, u) = \mathtt{RED}(n) \quad \Longrightarrow \quad |u| \geq |n|$.

Let us take: $L_0(M) = \{w \in \Sigma^*_M \mid \Delta^*_M(\text{«}\,w\,\text{»}) = \mathtt{ACC}\}$.

Constant. Let $k_M = \max\{\,|n|\,\big|\mathtt{RED}(n) \in F_M\,\}$. We call k_M the *characteristic constant* of M.

The operation of reduction. We will exactly describe a reduction of a word by a binary sequence with the help of the following operation $/$:

$a/\mathtt{0} = a, \quad a/\mathtt{1} = \lambda, \quad \lambda/n = \lambda, \quad u/\lambda = u,$

$(u \cdot a)/(n \cdot i) = (u/n) \cdot (a/i)$,

where $u \in \Sigma^*$, $a \in \Sigma$, $n \in (\mathtt{10}^*)^+$ and $i \in \{\mathtt{0}, \mathtt{1}\}$. The size of the strings u, n is here unbounded, moreover u can be longer then n, and vice versa. The reduction of the word $(\mathtt{a})$ by the sequence $\mathtt{1} \cdot \mathtt{0} \cdot \mathtt{1}$ is given in the following way: $(\mathtt{a})/\mathtt{101} = \mathtt{a}$.

Using just defined operation we can describe the way how the red-automaton M reduces a word $w \in \Sigma^*_M$.

The relation of reduction denoted by $\Rightarrow_M$ is introduced in the following way: $\text{«}w\text{»} \Rightarrow_M \text{«}w'\text{»}$,
if $\Delta^*_M(\text{«}w\text{»}) = \mathtt{RED}(n)$, and $\text{«}w\text{»}/n = \text{«}w'\text{»}$.

If $\text{«}w\text{»} \Rightarrow_M \text{«}w'\text{»}$ holds, we say, that the automaton M *reduces* the word w into the word w'. We can see that $|w| > |w'|$.

The relation $\Rightarrow^+$ is the transitive closure of $\Rightarrow$; $\Rightarrow^*$ is the reflexive and transitive closure of $\Rightarrow$.

Analysis by reduction by M is any sequence of reductions $\text{«}w_1\text{»} \Rightarrow \text{«}w_2\text{»} \Rightarrow \ldots \Rightarrow \text{«}w_n\text{»}$, which cannot be further prolonged. If $w_n \in L_0(M)$, we speak about *accepting* analysis by reduction, in the other case we speak about *rejecting* analysis by reduction. Often we will speak about analysis instead of analysis by reduction.

Stages. Let us recall that each computation of a red-automaton is divided in stages. At the beginning of each stage the head points at the leftmost item behind the left sentinel, and the control unit is in the (re)starting state. The stage ends as the control unit gets to any final state (operation) from F_M. There are three kinds of operations: $\mathtt{ACC}$, $\mathtt{ERR}$, and $\mathtt{RED}$. Accordingly, we have *accepting* ($\mathtt{ACC}$-), *rejecting* ($\mathtt{ERR}$-), and *reducing* ($\mathtt{RED}$-)*stages.*

Recognized language. The language *recognized* by M is defined in the following way:
$L(M) = \{\, w \mid «w» \Rightarrow_M^* «w'», \;\; and \;\; w' \in L_0(M) \,\}$.

Equivalences of red-automata. Two red-automata M_1 and M_2 are *equivalent*, if $L(M_1) = L(M_2)$.

We will often consider a stronger equivalence. Let us suppose that for any $w \in (\Sigma_{M_1}^* \cup \Sigma_{M_2}^*)\cdot\{\lambda, »\}$ hold at the same time

$\delta_{M_1}^*(«w) = \mathtt{ACC} \iff \delta_{M_2}^*(«w) = \mathtt{ACC}$,
$\delta_{M_1}^*(«w) = \mathtt{RED}(n) \iff \delta_{M_2}^*(«w) = \mathtt{RED}(n)$.

We say that M_1 and M_2 are *strongly equivalent.* We can see that if M_1 and M_2 are strongly equivalent then they are equivalent, as well.

Error and correctness preserving property. We can see the following usefull property:

Lemma 1. *If $«w_1» \Rightarrow_M «w_2»$, then $w_1 \in L(M)$, exactly if $w_2 \in L(M)$.*

Monotony. Monotony is an important property that enables to characterize the class of DCFL in terms of monotonic reducing automata. This property was introduced for restarting automata in [2], first. Informally a red-automaton M is monotonic if the size of sequences of non-visited items (symbols) in individual stages of any analysis by reduction by M is non-increasing. A monotonic reducing automaton will be called a *mon-red-automaton* for short. As an example see Table 1.

3 Robust analyzer

We introduce for the robust analysis by reduction a new type of automaton – *robust analyzer.* Robust analyzer enhances the reducing automaton by the ability of inserting special auxiliary symbols, and by the ability to read any input word about its input alphabet to the end. Robust analyzer A consists of finite control unit with a finite set of states S_A, and from a working head connecting the finite control unit with a linear list of items. The list of items is bounded by a left and right sentinels « and ». All other items contain a symbol from a finite *input alphabet* X_A, or of a finite *auxiliary alphabet* Y_A. These alphabets are mutually disjunct.

Robust analyzer is able to delete some items from the list (operation of reduction). Deleted can be the item visited by the working head and some items positioned not far to the left from the working head. Operations of reductions are controlled by reducing sequences. Each operation of reduction is followed by a restart, i.e., transfer of the control unit into the (re)starting state s_A, and a placement of the working head on the left sentinel «. It means, that the robust analyzer, similarly as reducing automaton, works in stages. Therefore we can define the monotony for robust analyzers in the same way as for reducing automata.

We divide auxiliary symbols into two types. Each type serves to a different purpose:
1) for a transfer of local informations between different stages,
2) for a marking of correct and incorrect sub-words of the analyzed word, and for marking of the place of certain types of a reducing conflict.

The auxiliary symbols of the type 2 are called *signs.* Here we use two signs:
`!` – for marking of incorrectnesses,
`?` – for marking of reducing ambiguities.
We understand under the reducing ambiguity a subword for which is obtained by the robust analysis an ambiguous information about its current reduction.

There is a technical difference between reducing automata, and robust parsers. The behavior of the robust parser is described by the following three functions:

Transition function $t_A : S_A \times (X_A \cup Y_A) \longrightarrow S_A$.
t_a determines the state, into which will be transfered the finite control from its current state after scanning the symbol from the item visited by the working head.

Inserting function $i_A : S_A \times (X_A \cup Y_A) \longrightarrow I_A$.
i_A assigns to a state, and to a scanned symbol an *inserting sequence* from a (final) set $I_A \subseteq (Y_A \cdot \mathtt{0}^*)^*$.

Inserting sequences serves in a similar way as reducing sequences. They describe the inserted auxiliary symbols, and their inserting positions. If the value of the inserting function is λ, it will be nothing inserted. If the value is `!`, A inserts new item with the marking `!` immediately before (to the left) the scanned item. The value `?000` says, that the marking `?` should be inserted before the third item to the left before the scanned item.

Reducing function $r_A : S_A \times (X_A \cup Y_A) \longrightarrow R_A$.
r_A assigns to a state, and to a scanned symbol a *reducing sequence* from a (final) set $R_A \subseteq (\mathtt{1} \cdot \mathtt{0}^*)^*$.

Reducing sequence is here interpreted in the same way as for reducing automata.

A step of the robust analyzer A consists from the sequence of the following actions:

Shift to the right. Analyzer A starts each step by a shift to the right of its working head to the next item of the working list.

Application of the inserting function. i_A by the current *situation* (state, symbol) determines the inserting sequence Is. A controlled by Is inserts new auxiliary symbols.

Application of the reduction function. r_A determines by the current situation the current reducing sequence Rs. If RS is non-empty, A controlled by Rs reduces (deletes) determined items from the working list. After such a reduction A finishes the step by a restart, i.e., moves the working head on the left sentinel «, and transfers the control unit into the (re)starting state s_A. Then a new stage will be started on the reduced list.

If Rs is empty then A does not perform any reduction, neither the restart, and it finishes the step by the following action.

Application of the transition function. t_A determines by the current situation the new state q. Then A continues from the state q by a further step of the current stage.

The last difference of A from reducing automata consists in the fact that A contains only one halting state $\texttt{END} \in S_A$. We will see that the signs of A will refine the ability of accepting and rejecting by the states ACC and ERR of reducing automata. For this purpose we observe the signs `!` and `?` inserted in the different stages of the computation into the gradually reduced working list. We will project the signs into the original input list in such a way that we will insert the signs into the same positions (i.e. before the same items) into which they were inserted during the individual stages of the computation (robust analysis) by A.

We denote by $p_A(w)$ a word w enriched by the signs (in the way mentioned above) inserted into the list during the analysis by the robust analyzer A. We will later formulate our results using this denotation. As an example see Fig.1. We can consider $p_A(w)$ as the output word of A.

3.1 Prefix and post-prefix (in)consistencies

Assumption. We assume in the following that $L \subseteq \Sigma^*$, and any symbol of Σ is a symbol of some word from L.

We call a word v *inconsistent (incorrect)* with respect to the language $L \subseteq \Sigma^*$, if for any u, $w \in \Sigma^*$ is $uvw \notin L$.

We can see that incorrect words can obtain proper incorrect sub-words. This fact lead us to the following notion.

We say that a word v is an *incorrect core* of the word w with respect to the language L, if it is a subword of w, if it is incorrect with respect to the language L, and if it is minimal by the ordering "to be a sub-word".

On the other hand, a word v is a *correct* sub-word of a word w with respect to the language L, if $w = xvy$, and for some x', y' is $x'vy' \in L$. We say that v is a *correct core* of a word w with respect to the language L, if it is a correct sub-word of w with respect to L, and it is maximal by the ordering "o be a sub-word". The assumption that each symbol of Σ is a symbol of some word of the language L ensures that each symbol of any word $w \in \Sigma^*$ is contained in some correct core of this word.

Prefix consistence is the longest correct prefix v of the analyzed word w. *Prefix inconsistence* is the shortest incorrect prefix of the analyzed word w, i.e., it is the prefix va of w, where $a \in \Sigma$. *Post-prefix consistence* is a suffix x of a correct core behind (to the right of) the prefix consistence, or behind some of the previous post-prefix consistencies. We assign to the post-prefix consistency x the incorrect sub-word xa of w. We say that xa is a post-prefix inconsistence of w (with respect to L).

Our effort in the following is to deterministically, in a monotonic way, and exactly to localize the prefix and post-prefix (in)consistencies in the analyzed words from DCFL.

3.2 Post-prefix robust analyzer *A*

Prefix consistence. A red-automaton M is *prefix-consistent* when for each word u and each symbol a (including the right sentinel) it holds the following: if $\Delta_M(s_M, u) \in S_M$ and $\Delta_M(s_M, ua) \neq \texttt{ERR}$, then ua is a prefix of some word from $L(M) \cdot \{»\}$.

The following proposition is derived from the main result from [2]. The detailed proof is in [6]. There is also connected with some other propositions.

Proposition 1. *Monotone, prefix consistent, red-automata characterize the class of DCFL.*

It is shown in [7] that the notion of red-automata is useful for the techniques of minimization. There is to any red-automaton M constructed an unambiguously determined state-minimal red-automaton M_m which is strongly equivalent with M.

We will show informally in the next part a method how construct for a given monotone, prefix-correct, state-minimal reducing automaton M a robust analyzer A which determines in any word $w \in «\Sigma_M^*»$ the prefix-(in)consistence, and (not obligatory all) post-prefix (in)consistencies. We suppose for the construction that $L(M) \neq \emptyset$.

At first A will use the prefix-consistency of M for the finding of the prefix-(in)consistency of the analyzed word w.

Such a situation can occur after one, or after more stages if M will be transfered into the final rejecting state $\texttt{ERR}$. The computation (analysis) of M on the word w until this moment we describe in the following way:

1) At first M (possibly) gradually reduces the word w into the word w', i.e., $w \Rightarrow_M^* w'$.

2) Then in the next stage M transfers over some prefix x of the word w' into some non-final state $s \in S_M$, i.e., $\delta_M^*(s_M, x) = s \in S_M$,

3) Finally from the state s transfers over the next symbol a into the final state ERR, i.e., $\delta_M(s,a) = \texttt{ERR}$.

We can see that A has founded by the previous simulation of M the prefix inconsistency of w. For marking of the prefix inconsistency A inserts the sign **!** between the correct prefix «x and the symbol a.

The prefix-consistency of M ensures that M has visited in the last step described above the symbol a at the first time. Therefore if $w' =$ «xay for some y then ay is a suffix of the original input word w.

Let us now informaly describe how A continues in the robust analysis over the mentioned suffix ay of the word w.

We will use the function δ_M for this aim. This function was introduced as an enhancement of the transition function f_M. It describes not only the transfers between the individual states, but also the tranfers between the indiviual subsets of the set $S_M \cup F_M \cup \{\texttt{RED}\}$, i.e., of the set of all final and non-final states, and of a special state RED. We will use it in the following in order to describe the all possible (partial) computation of M over the suffix ay at the same time.

We let A to compute the function δ_M over the suffix $ay = a_0a_1 \dots a_{|y|}$ starting from the set S_M of all non-final states of M. A will control the computation in the following way. Let us initially take the set S_M as a set further denoted as S_I.

Let us denote the following part of the computation of A as a cycle C_1. The cycle C_1 is performed until for the set $S = \delta^*_M(S_I, a_0 \dots a_i)$, where $0 \leq i \leq |y|$, holds that $\emptyset \subset S \subseteq S_M \cup \{ERR\}$, and S contains some non-final state. Then A performs the following action: the head of A will be placed to the next item to the right, and as (the current value of) the set S will be taken the set $\delta_M(S, a_{i+1})$. Here ends the description of C_1.

The core of the post-prefix analysis by A are the following four cases where is not fulfilled the condition for the continuation of the computation by cycle C_1.

Correct suffix. The set S contains the accepting state ACC; i.e., $\texttt{ACC} \in S$.
If $\texttt{ACC} \in S$, then the current suffix of the analyzed word w by A is a suffix of some word from $L(M)$. Therefore, the current suffix cannot contain any further inconsistency. The work of A on w is finished at this moment.

An unambiguous inconsistency. The set S contains a single state – the rejecting state ERR; i.e., $S = \{\texttt{ERR}\}$.

All the possible computations of M over the word w behind the previous inconsistency has ended at the same time in the state ERR. We have found a suffix of a correct core of the analyzed word, i.e., one of its post-prefix (in)consistency. At this moment A inserts the sign **!** immediately before the position of its working head. The automaton A will look for a new post-prefix (in)consistency behind (to the right from) the currently inserted sign **!**. A will take instead of the set $\{\texttt{ERR}\}$ as the current value of the set S the set $\delta_M(S_M, a)$, where a is the symbol scanned by the working head. A will continue in the robust analysis of the remaining suffix by the schema of the cycle C_1.

An ambiguous reduction. S does not contain ACC, and either does contain two different reducing states of M, or does contain at least one non-final state, and at least one reducing state; i.e.,
$\texttt{ACC} \notin S, \quad \text{and} \quad \exists n : \texttt{RED}(n) \in S \not\subseteq \{\texttt{RED}(n), \texttt{ERR}\}.$
We say that S fulfilling the condition above is an *ambiguous* set.
The task for A is to work without false inconsistency messages. From that reason A separates the ambiguous part from the remaining suffix. It inserts the sign for the ambiguity **?** in the place of the current ambiguity, i.e., immediately to the left from the position of the working head (if the sign is not already placed there in some of the previous stages). At this moment A takes for the set S the complete set S_M, and continues in the robust analysis behind the sign **?** by the scheme of the cycle C_1.

An unambiguous reduction. The set S contains exactly one reducing operation, and possibly beside it the final state ERR; i.e., $\exists n : \texttt{RED}(n) \in S \subseteq \{\texttt{RED}(n), \texttt{ERR}\}$.

Let us denote as u the sub-word which is created by the input symbols positioned between the last sign **!** or **?**, and the position of the working head including the scanned symbol. The sub-word u is because of the prefix-consistency, and because of the state minimality of M a sub-word of some word from $L(M)$. Moreover, the u is reduced in any word $w \in L(M)$ of the form $w = vux$ by the reducing seguence n, i.e, the reducing sequence and the position of the reduction are determined unambiguously. A will reduce also by n, but only the symbols from u if we consider the case that n can be longer then u.

Observation. The reducing sequence n deletes at least one symbol from u. This observation follows from the unambiguity of the reduction of u. Let us suppose the opposite. Then for some z, and n' is $n = n' \cdot \texttt{0}^{|u|}$, and $\delta^*_M(s_M, zu) = \texttt{RED}(n)$.
Let z' be the shortest z with the properties described above. Since M is prefix-correct «$z'u$ is a prefix of some word from «$L(M)$». In the next stage occurs one of the following variant:

$$\begin{aligned}
&\delta^*_M(s_M, (z'/n') \cdot u) = \texttt{RED}(n'') \quad \text{for some } n'' \\
&\delta^*_M(s_M, (z'/n') \cdot u) = \texttt{RED} \\
&\delta^*_M(s_M, (z'/n') \cdot u) = \texttt{ACC} \\
&\delta^*_M(s_M, (z'/n') \cdot u) \in S_M
\end{aligned}$$

Each of the presented variant leads to a contradiction with the unambiguity of the reduction of the word u. If occurs the first one, then n'' reduces some symbol of the sub-word u (since z' cannot be shorter), therefore $n'' \neq n$ and $\mathtt{RED}(n'') \in S \not\subseteq \{\mathtt{RED}(n), \mathtt{ERR}\}$. By the remaining variants is the contradiction obvious.

Apart from the reduction of u by n, A will insert into the list a new item with an auxiliary symbol – the set U of pairs of an internal state and a word over an input alphabet of the length k_M at most. This auxiliary symbol will be used in the next stage to adjust the set of states computed by the function δ_M. Our goal is to avoid situation when $\delta^*_M(s,u) = \mathtt{ERR} \neq \delta^*_M(s,u/n)$ for some $s \in S_M$. Such internal states s must be eliminated. The set U is defined by the following way:

- If $|u| \geq |n|$, then A reduces the working list of items by n and A puts a new item with the auxiliary symbol U just in front of the leftmost deleted item. $U = \{(s,\lambda) \mid \exists s' \in S_M : \delta^*_M(s',u_1) = s$ and $\delta^*_M(s,u_2) = \mathtt{RED}(n)\}$ where $u = u_1u_2$ and $|u_2| = |n|$.
- If $|u| < |n|$, then A cannot reduce by the whole n as such a reduction would impact a part of the working list in front of u; this part would be reduced by n_1 such that $n = n_1n_2$ and $|n_2| = |u|$. But a part of the working list in front of the rightmost marker `!` or `?` can be reduced in some word of $L(M)$ in other way or even not at all. So, A will reduce items behind the rightmost marker `!` or `?` by the reducing sequence n_2 and it will insert a new item containing an auxiliary symbol U just to the right of the rightmost marker. $U = \{(s,x) \mid \exists v \in \Sigma^*_M : x = v/n_1$ a $|v| = |n_1|$ a $\delta^*_M(s,vu) = \mathtt{RED}(n)\}$.

Insertion of the set U into the working list is important as it ensures the continuity of subsequent stages of computation. In next stage, A will use this set to adjust the set S of internal states computed by function δ_M. As soon as A reach the item with U, it substitute S by $S' = \{\delta^*_M(s,v) \mid (s,v) \in U\}$. It guarantees that A enter a part of the list impacted by the last reduction in such states only that led to the last reduction of u by n resp. n_2.

A uses just defined set U in such a case only when the new item with U is inserted just behind an item containing a symbol of the input alphabet or a marker. Otherwise, when this item contains an auxiliary symbol U' different from both markers, then (instead of insertion of U) A replaces U' with U computed in the following way:

$$U = \{(s,x) \mid \exists y, (s',x') \in U' : \ x = yx'/n_1 \text{ , and } \\ \delta^*_M(s,y) = s' \text{ , and } \delta^*_M(s',x'u) = \mathtt{RED}(n) \text{ , and } \\ |y| = \max\{0, |n_1| - |x'|\}\},$$

where n_1 is a prefix of n of the length $|n| - |u|$. In all cases, the length of the word x contained in any pair of inserted set U is bounded by the characteristic constant k_M which ensures that U is finite.

Let us note that A stores in its finite control a suitable suffix of its working list before the position of its working head. The length of this suffix need not be longer then $2 \cdot k_M$. It contains the input items, and the inserted values of the set S in the last $2 \cdot k_M$ steps.

Recall, that we suppose that the automaton M is minimal and prefix-consistent. The minimality of M ensures that each state of M is reachable. The state-reachability, and the prefix-consistence of the automaton M ensure for any word u, that u is a sub-word of some word of $L(M)$, if $\delta^*_M(S_M,u) \not\subseteq \{\mathtt{ERR}, \mathtt{RED}\}$.

If A inserts the sign `!` or `?` immediately before the right sentinel » it finishes its computation. Since we suppose that $L(M)$ is non-empty, is » a suffix of some word of the language «$L(M)$» (i.e.,$L(M)$ with sentinels).

Now we have outlined the behavior of A in the first stage after the localization of the prefix-inconsistence. In the next stages we need also to consider the signs and the other auxiliary symbols inserted in the previous stages. We will not describe here these details.

We illustrate the outlined method by the following example.

Example 1. We explain the presented method by two prefix-consistent, state-minimal mon-red-automata. We will present two different robust analyses of the following inconsistent word «`a` + +(`a`)+)(`a`».

The transition functions of the automata M_1 and M_3 from [6] which we use in this example are defined by the tables in the figure 1. We do not present here the automaton M_2 from [6].

The robust reduction analyses of the considered word are on figures 1a, and 1b. Both figures contain also the input word enriched by the signs `!` and `?` on the corresponding places.

Let us note that in [6] is a detailed description of a construction which constructs to a given prefix consistent, state minimal mon-red-automaton M its robust analyzer A. This construction implements the outlined method. The robust analyzer A is by this construction given unambiguously for a given M.

3.3 Guarantees of the presented method

We formulate the guarantees of the presented method as theorems. The detailed proofs can be found in [6].

Theorem 1. *Let M be a prefix consistent, state-minimal mon-red-automaton and A its post-prefix robust analyzer. For any $w \in$ «Σ^*» the following propositions hold:*

1. *The analyzer A reads (in one or more stages) the complete word «w» and finishes its computation in a special state* END. *Any computation of A is monotone.*
2. *If p_A(«w») does not contain any sign* **!**, *then p_A(«w») = «w» and w is from $L(M)$.*
3. *If «w» ∈ «$L(M)$» then p_A(«w») = «w».*
4. *If «u***!** *is a prefix of p_A(«w») and u does not contain the sign* **!** *then u does not contain the sign* **?** *as well, and «u is the longest correct-prefix of the word «w» with respect to the language «$L(M)$».*
5. *If* **!***u***!** *or* **?***u***!** *is a sub-word of the word p_A(«w») and u does not contain any sign* **!** *or* **?** *then u is a suffix of some corect core of the word «w» with respect to the language «$L(M)$».*
6. *If* **!***u» or* **?***u» is a suffix of the word p_A(«w»), and u does not contain any sign* **!** *or* **?** *then u» is a sufix of some word from «$L(M)$».*
7. *If* **!***u***?** *or* **?***u***?** *is a sub-word of p_A(«w») and u does not contain any sign* **!** *or* **?** *then u is a sub-word of some word from «$L(M)$».*

	a	+	(	)	»
⇒ s_0	s_1	ERR	s_2	ERR	ERR
s_1	ERR	RED(11)	ERR	ERR	ACC
s_2	s_3	ERR	s_2	ERR	ERR
s_3	ERR	s_4	ERR	RED(101)	ERR
s_4	s_5	ERR	s_2	ERR	ERR
s_5	ERR	s_4	ERR	RED(110)	ERR

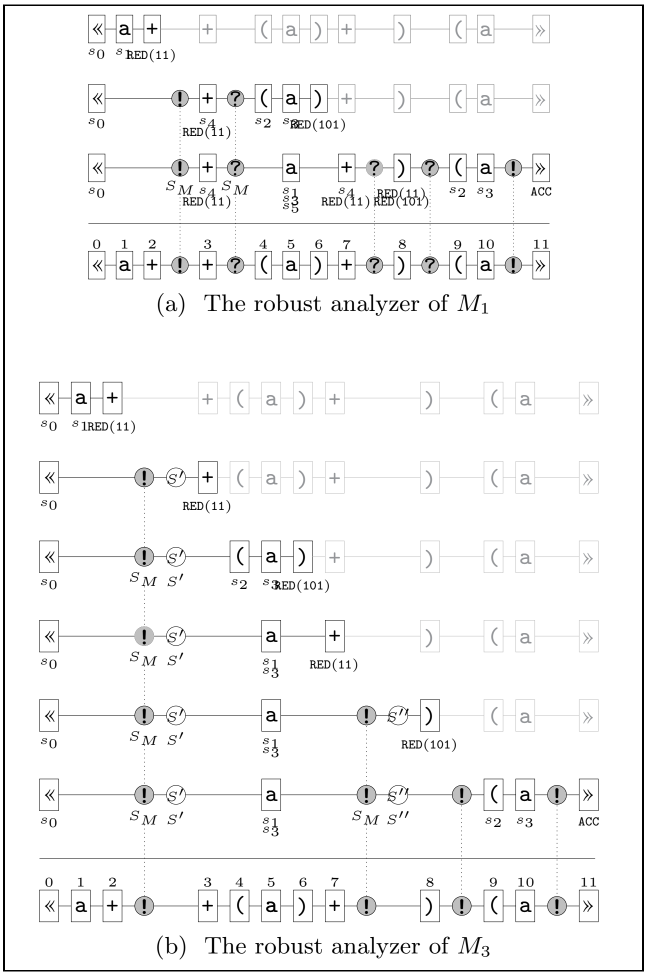
(a) automaton M_1 reducing the word a+ without brackets around it "from the left" and the +a with brackets around "from the right"

	a	+	(	)	»
⇒ s_0	s_1	ERR	s_2	ERR	ERR
s_1	ERR	RED(11)	ERR	ERR	ACC
s_2	s_3	ERR	s_2	ERR	ERR
s_3	ERR	RED(11)	ERR	RED(101)	ERR

(b) (strongly) monotone automaton M_3 reducing a+ only "from the left"

Table 1: The transition functions for M_1 and M_3.

We will discuss the meaning of the sign ?, and we will formulate properties of the mon-red-automaton M which ensure that its post-prefix robust analyzer A does not use the sign ? at all. This sign serves as the right sentinel for the correct sub-words of $L(M)$ which lead A to some of two following types of a reduction conflict. Let u be such a sub-word which is followed by ?. The first type of the conflict means that there are two different words of $L(M)$ containing u which lead M by reading the complete u and its prefixes to two different reductions. The second type of the conflict means that there is a transfer trough u by M which leads to a reduction, and at the same time there is an another transfer leading to the shift to the right of M from u to the next symbol.



« a + + (a) +) (a »
s_0 s_1 RED(11)
! ? RED(11) s_2 s_3 RED(101)
S_M s_4 S_M RED(11) RED(101) ACC
0 1 2 3 4 5 6 7 8 9 10 11
« a + ! + ? (a) + ?) ? (a ! »
S' S''
« a + ! + (a) + !) ! (a ! »

(a) The robust analyzer of M_1

(b) The robust analyzer of M_3

Fig. 1: The robust analysis of «a++(a)+)(a».

Now we gradually introduce the notions of *unambiguously reducible sub-word* and *unambiguously reducing red-automaton*, and we will show that a post-prefix robust analyzer A of M which is unambiguously reducing, need not to use the sign ? at all.

We say that a sub-word w is *reducible* by M if for some n holds that $\mathtt{RED}(n) \in \delta^*_M(S_M, w)$.

We say that a sub-word w is *unambiguously reducible* by M if for some n holds that $\mathtt{RED}(n) \in \delta^*_M(S_M, w) \subseteq \{\mathtt{RED}(n), \mathtt{ERR}\}$.

We say that a sub-word which is reducible but it is not unambiguously reducible is an *ambiguously reducible* sub-word.

We say that M is *unambiguously reducing* if any of its reducible sub-words is unambiguously reducible.

Example 2. The automaton M_1 from the example 1 is not unambiguously reducing. All its ambiguously reducible sub-words are presented in Table 2a. Let us note the length of this words is not limited by any constant, since for any $i \geq 0$ holds that

$\delta^*_{M_1}(S_{M_1}, (\texttt{+a})^i\texttt{+}) = \{\texttt{ERR}, \texttt{RED(110)}, s_4\}$.

The minimal unambiguously reducible sub-words of M_1 are in Table 2b.

sub-word w	`)`	`a)`	`+`	`a+`	`+a+`	...

$\delta^*_{M_1}(S_{M_1}, w)$	RED(110)
	RED(101)
		RED(110)
		RED(101)
			RED(11)
			s_4
				RED(11)
				s_4
					RED
					s_4
						...

(a) ambiguously reducible sub-words

sub-word w	`+a)`	`(a)`	`«a+`
$\delta^*_{M_1}(S_{M_1}, w)$	RED(110)	RED(101)	RED(11)

(b) minimal unambiguously reducible sub-words

Table 2: Reducible sub-words of M_1.

We can see that in this example the robust analyzer of M_3 has founded all inconsistencies in the word «$\texttt{a} + +(\texttt{a})+)(\texttt{a}$»., i.e., it has not used the sign ? at all. This example illustrates the fact that M_3 is an unambiguously reducing red-automaton. We can see that directly from Table 1b.

The meaning of the reducing unambiguity for the localization of the post-pefix inconsistencies summarizes the following theorem.

Theorem 2. *Let M be a prefix consistent, state-minimal mon-red-automaton and A its post-prefix robust analyzer. If M is at the same time unambiguously reducing then its post-prefix robust analyzer A does not use the sign* ? *at all. That means, that A in any word from «Σ^*_M» determines the prefix inconsistency and all its post-prefix inconsistencies with respect to the language $L(M)$.*

Corollary 1. *Let M be a mon-red-automaton which is at the same time prefix-consistent and state-minimal, and A be its robust analyzer. Then there is a deterministic push-down transducer which translates any word w from Σ^*_M on the word $p_A(w)$.*

4 Conclusion

The presented method can be considered as a direct generalization of the method presented in [9], and as an essential refinement and a generalization of the method from [5]. The method in [9] is based on monotone reducing automata, the method in [5] is based on (monotone) list automata with auxiliary symbols. Both the methods are based on the so called head-symbols. The head-symbol (in)consistencies from [9] create a very special type of (in)consistencies considered by the method presented in this paper. In the close future we will show that the set of languages recognized by unambiguously reducing, prefix consistent, state-minimal mon-red-automata creates a proper subclass of DCFL.

References

1. G. V. Cormack: *An LR substring parser for noncorrecting syntax error recovery.* In: Proc. of PLDI '89, 1989, 161–169.
2. P. Jančar, F. Mráz, M. Plátek, J. Vogel: *Restarting automata.* In Proc. FCT'95, Dresden, Germany, August 1995, LNCS 965, Springer Verlag 1995, 283–292.
3. F. Otto: *Restarting automata.* In: Z. Ésik, C. Martin-Vide, and V. Mitrana (Eds.), Recent Advances in Formal Languages and Applications, Studies in Computational Intelligence, Vol. 25, Springer, Berlin, 2006, 269–303.
4. Gh. Păun: Marcus Contextual Grammars, Kluwer, Dordrecht, Boston, London, 1997.
5. M. Plátek: *Construction of a robust parser from a deterministic reduced parser.* Kybernetika 33(3), 1997, 311–332.
6. M. Procházka: *Redukční automaty a syntaktické chyby.* (in Czech) Text for PhD Dissertation, it will be to achieve soon on the web.
7. M. Procházka: *Concepts of syntax error recovery for monotonic reducing automata.* MIS 2004, 94–103, http://ulita.ms.mff.cuni.cz/pub/MIS/MIS2004.pdf
8. M. Procházka, M. Plátek: *Redukční automaty – monotonie a redukovanost.* ITAT 2002, 23–32.
9. M. Procházka, M. Plátek: *Redukční automaty a syntaktické chyby.* Proceedings of ITAT 2011 (in Czech), 2011, 23–30.

ISBN 978-80-971144-0-4

9 788097 114404 >