Jan Kofroň and Tomáš Vojnar (Eds.)

MEMICS 2015

Tenth Doctoral Workshop on Mathematical and Engineering Methods in Computer Science

Telč, Czech Republic, October 23–25, 2015
Preface

This volume contains the proceedings of the 10th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS 2015) held in Telč, Czech Republic, during October 23–25, 2015.

The aim of the MEMICS workshop series is to provide an opportunity for PhD students to present and discuss their work in an international environment. The scope of MEMICS is broad and covers many fields of computer science and engineering. In the year 2015, we paid a special attention to submissions in the following (though not exclusive) areas:

- Security and safety
- Bioinformatics
- Recommender systems
- High-performance and cloud computing
- Non-traditional computational models (quantum computing, etc.)

There were 25 submissions from PhD students from 9 countries. Each submission was thoroughly evaluated by at least three Program Committee members who also provided extensive feedback to the authors. Out of these submissions, 10 full papers were selected for publication in LNCS post-proceedings, and additional 7 papers were selected for publication in these local proceedings.

In addition to regular papers, MEMICS traditionally invites students to submit a presentation abstract of their recent already published work. This work should be presented or accepted at a high-quality conference or in a recognized journal. A total of 14 presentation abstracts were submitted from 11 countries out of which 12 were included into the MEMICS 2015 program.

All of the contributed papers were presented by PhD students who received immediate feedback from their peers and the participating senior researchers. All students were encouraged to actively take part in the discussions, express their opinions, exchange ideas and compare methods, traditions and approaches between groups and institutions whose representatives were participating in the workshop.

The highlights of the MEMICS 2015 program included six keynote lectures delivered by internationally recognized researchers from the areas of interest. The speakers were:

Ezio Bartocci (Vienna University of Technology, Austria)  
Siegried Benkner (University of Vienna, Austria)  
Mike Just (Heriot-Watt University, Edinburgh, Scotland)  
Simone Severini (University College London, United Kingdom)  
Natasha Sharygina (University of Lugano, Switzerland)  
Peter Vojtšáš (Charles University Prague, Czech Republic)

The full papers of these keynote lectures are also included in the LNCS post-proceedings.
The MEMICS tradition of best paper awards continued also in the year 2015. The best contributed paper was selected during the workshop, taking into account its scientific and technical contribution together with the quality of presentation. The award consisted of a diploma accompanied by a financial prize of roughly 400 Euro. The prize money was donated by ZONER software, a. s.

The successful organization of MEMICS 2015 would not be possible without generous help and support from the organizing institutions: Brno University of Technology, Masaryk University in Brno, and Charles University Prague.

We thank the Program Committee members and the external reviewers for their careful and constructive work. We thank the Organizing Committee members who helped to create a unique and relaxed atmosphere which distinguishes MEMICS from other computer science meetings. Our thanks go in particular to the organization chair Radek Kočí and to Jaroslav Rozman for creating the proceedings.

We also gratefully acknowledge the support of the EasyChair system and the great cooperation with the Lecture Notes in Computer Science team of Springer Verlag.

Prague, September 2015

Jan Kofroň  
*PC chair of MEMICS’15*

Tomáš Vojnar  
*General chair of MEMICS’15*
Organisation

General Chair

Tomáš Vojnar, Brno University of Technology, Brno, Czech Republic

Programme Committee Chair

Jan Kofron, Charles University, Czech Republic

Programme Committee

Andris Ambainis, University of Latvia, Latvia
Gianni Antichi, University of Cambridge, UK
Michal Baczyński, Uniwersytet Śląski w Katowicach, Poland
Ezio Bartocci, Vienna University of Technology, Austria
Jan Bonda, Masaryk University, Czech Republic
Krishnendu Chatterjee, IST Austria, Austria
Markus Chimani, Osnabrück University, Germany
Pavel Čeleta, Masaryk University, Czech Republic
Eva Dokládalová, ESIEE Paris, France
Martin Drahanský, Brno University of Technology, Czech Republic
Frédéric Dupuis, Masaryk University, Czech Republic
Jiří Filipovič, Masaryk University, Czech Republic
Vojtěch Forejt, University of Oxford, UK
Dieter Gollmann, TU Hamburg, Germany
Derek Groen, University College London, UK
Petr Hanáček, Brno University of Technology, Czech Republic
Lucía Happe, KIT, Germany
Marcus Huber, Universitat Autònoma de Barcelona, Spain
Antti Hyvärinen, University of Lugano, Switzerland
Ondřej Jakl, VŠB-TU Ostrava, Czech Republic
Jiří Jaroš, Brno University of Technology, Czech Republic
Stefan Kiefer, University of Oxford, UK
Łukasz Kowalik, University of Warsaw, Poland
Stanislav Krajič, Pavol Jozef Šafárik University in Košice, Slovak Republic
Dieter Kranzlmüller, Ludwig-Maxmillian Univeristy and also Supercomputing Centre Garching Munich, Germany
Daniel Kráľ, University of Wawrick, UK
Daniel Langr, Czech Technical University in Prague, Czech Republic
Luděk Matyska, Masaryk University, Czech Republic
Manuel Ojeda Aciego, University of Malaga, Spain
Table of Contents

Preface ....................................................................................... V
Organisation ............................................................................. VII

I Invited Lectures – Abstracts

II Contributed Papers – Abstracts

III Regular Papers

Devectorization of Array Expressions for Selective Just in Time Specialization of Neural Ensamble Simulations ......................... 15
   Jan Fousek

Parallel Computations of Differential Equations .......................... 28
   Filip Kocina, Petr Vajgner, Gabriela Nečasová, and Jiří Kunovský

Absolutely Unlimited Deep Pushdown Automata .......................... 36
   Jiří Kučera, Alexander Meduna, and Ondřej Soukup

Improving Sketches for Similarity Search .................................. 45
   Vladimír Mic, David Novak, and Pavel Zezula

ECHO NGS Error Correction Reimplemented: Huge Speedup for Little Loss of Accuracy ......................................................... 58
   Miloš Šimek, Aleš Krček, Jan Ištvanek, Jana Řepková, and Radka Svobodová Vařková

Iterative Optimization of a Multi-compartmental Air Quality Modelling System ................................................................. 70
   Martin Tomáš, Tomáš Rebok, and Christos Efthathioú

IV Presentations

Randomized fuzzy formal contexts .............................................. 85
   Lubomír Antoni, Stanislav Krajčí, and Ondrej Krídlo
The Hanoi Omega-Automata Format ............................................. 86
Tomáš Babiak, František Blahoudek, Alexandre Duret-Lutz, Joachim Klein, Jan Křížinský, David Müller, David Parker, and Jan Strejček

Device-independent randomness extraction from an arbitrarily weak min-entropy source ...................................................... 88
Jan Bouda, Marcin Pawlowski, Matej Pivoluska, and Martin Plesch

Nested Antichains for WS1S......................................................... 89
Tomáš Fiedor, Lukáš Holík, Ondřej Lengál, and Tomáš Vojnar

Optimizing CUDA code by kernel fusion: application on BLAS ........... 90
Jiří Filipovič, Matus Madzin, Jan Fousek, and Luděk Matyska

FO Model Checking on Posets of Bounded Width .......................... 91
Jakub Gajarský

Search-Space Partitioning for Parallelizing SMT Solvers .................. 92
Antti E. J. Hyvärinen, Matteo Marescotti, and Natasha Sharygina

Large-scale Ultrasound Simulations Using the Hybrid OpenMP/MPI Decomposition ......................................................... 94
Jiří Jaroší, Vojtech Nikl, and Bradley E. Treeby

Characterizing multipartite entanglement without shared reference frames 95
Claude Klöckl, and Marcus Huber

Optimizing Job Scheduling in National Grid Computing System:
Theory and Practice ............................................................... 96
Dalibor Klusáček, Šimon Tóth, and Gabriela Podolníková

Toward natural multi-user interaction in advanced collaborative display environments ...................................................... 97
Vít Rusnák, Lukáš Račka, and Petr Holab

Optimized SLA Assured Service Brokering (SLaB) and Service Verification in Multi-Cloud Environment ................................. 98
Shyam S. Wagle

Author Index ........................................................................... 99
Part I

Invited Lectures – Abstracts
Ezio Bartocci, Vienna University of Technology, Austria
Formal Methods for Monitoring and Synthesis of Spatio-Temporal Properties

Networked dynamical systems are increasingly used as models for a variety of processes ranging from robotic teams to collections of genetically engineered living cells. As the complexity of these systems increases, so does the range of emergent properties that they exhibit. In pattern recognition and machine learning, spatio-temporal properties are usually classified according to statistical descriptors (or features). This approach despite its success and popularity lacks a rigorous foundation to specify such patterns and to reason about them in a systematic way. On the other end, formal methods provide logic-based languages with a well-defined syntax and semantics to specify in a precise and concise way emergent behaviours and the necessary techniques to automatically detect them. Formal analysis is however challenging due to the difficulty of defining spatio-temporal properties in a suitable formal language. In this talk, we will present some recent results on novel formal logic-based languages and methods to monitor and synthesize spatio-temporal properties in networked dynamical systems.

Siegried Benkner, Faculty of Computer Science, University of Vienna
Programming Support for Future Parallel Architectures

Due to physical constraints the performance of single processors has reached its limits, and all major hardware vendors switched to multi-core architectures. In addition, there is a trend towards heterogeneous parallel systems comprised of conventional multi-core CPUs, GPUs, and other types of accelerators, mainly because such systems promise a better performance/power ratio. As a consequence, the development of applications that can exploit the full potential of emerging parallel systems is becoming more and more challenging. In this talk, we outline the major challenges associated with software development for future parallel architectures, including higher-level parallel programming models, advanced runtime technology, and automatic performance tuning tools. In particular, we will present a high-level compositional approach to parallel software development in concert with an intelligent runtime system and automatic performance tuning techniques. Such an approach can significantly enhance programmability of future parallel systems, while ensuring efficiency and performance portability across a range of different architectures. We report on recent research results of two European projects, PEPPER and AUTOTUNE, which addressed the challenges of software development for current and emerging parallel systems, and discuss related research efforts and potential future directions.

Mike Just, Heriot-Watt University, Edinburgh, Scotland
Recent trends in authentication research

The past five years have been especially busy for the study of authentication techniques, such as passwords. In this talk, I will review several recent tech-
niques for improving authentication security and usability, including the use of honey passwords and renewed investigations into the use of randomly generated passwords. In addition, I will discuss the evolution of metrics that are used for measuring password security, including entropy and more recent attempts to model the guessing patterns of attackers.

Simone Severini, University College London
Graphs and quanta

I will talk about various ways to associate a graph to a quantum mechanical state or a quantum operation. This is an old story but still a good excuse to use quantum theory to play with graphs and graphs to play with quantum theory.

Natasha Sharygina, University of Lugano, Switzerland
Flexible interpolation for efficient model checking

Craig interpolation is widely used in symbolic model checking. Most interpolation algorithms construct the interpolant from a proof of unsatisfiability and a fixed labeling determined by which part of the propositional formula is being over-approximated. While this results in interpolants with fixed strength, it limits the possibility of generating interpolants of small size. This is problematic since the interpolant size is a determining factor in achieving good overall performance in model checking. This talk presents a new framework with a flexible labeling approach to construct small interpolants. In addition to providing the labeling mechanism guaranteeing small interpolants, this framework is capable to manage the strength of the interpolants. This talk presents the flexible interpolation mechanism in action by demonstrating how it works in incremental model checking. We will discuss two different applications: verification of multiple properties of the same code and verification of software upgrades. Both use interpolation as a backbone technology and face different strength requirements. We will demonstrate how flexible interpolation outperforms any of the fixed interpolation algorithms.

Peter Vojtáš, M. Vomlelová, M. Kopecký, Charles University, Czech Republic
Understanding transparent and complicated users as instances of preference learning for recommender systems

In this talk we are concerned with user understanding in content based recommendation. We assume we have a user-item matrix with explicit rating and a time stamp. Our task is to learn/predict user preference on unseen/unrated items. In this talk we are not going to present new mining techniques nor improve achievements of comparable efforts on some data and metrics. Rather, we concentrate on three issues: First, we study three different data sets from the same domain of movie recommendation, trying to avoid features specific for single
data and try to be more generic. We consider also semantic enrichment of movie data to enable content based recommendation. Second, we use several metrics which were not used so far in the recommender systems domain. Besides classical rating approximation with RMSE and top-k we study new metrics for predicting next-k and (at least) k-hit. We consider those parameters k variable and try to measure their behavior. Finally and most importantly, we trace performance of our methods and metrics as distribution along each single user. This helps us to detect transparent and complicated users. Improving these metrics (for instance 1-hit) can contribute to the increase of user satisfaction. We provide results of experiments with several combination of methods, data sets and metrics along of these three axes. Nevertheless, all our experiments are offline on public accessible data. A validation in real online A/B testing needs access to an application. We were not able to realize this so far. We concentrate only on algorithmic part of recommendation, business and marketing part of recommendation problem is out of scope of this study.
Part II

Contributed Papers – Abstracts
Span-program-based quantum algorithms for graph bipartiteness and connectivity

Span program is a linear-algebraic model of computation which can be used to design quantum algorithms. For any Boolean function there exists a span program that leads to a quantum algorithm with optimal quantum query complexity. In general, finding such span programs is not an easy task. In this work, given a query access to the adjacency matrix of a simple graph $G$ with $n$ vertices, we provide two new span-program-based quantum algorithms: - an algorithm for testing if the graph is bipartite that uses $O(n\sqrt{n})$ quantum queries; - an algorithm for testing if the graph is connected that uses $O(n\sqrt{n})$ quantum queries.

Fitting aggregation operators

This paper treats the problem of fitting aggregation operators to empirical data. Specifically, we are interested in modelling of the conjunction in human language. To our knowledge, the first attempt to see how humans "interpret" the conjunction for graded properties is due to the paper [1]. In that case, simply the minimum t-norm came out. Our results are different because our approach to the resolution is different. We have experimentally rated simple statements and their conjunctions. Then we have tried, on the basis of measured data, to find a suitable function, which corresponds to human conjunction. First, we discuss methods applicable to associative operators, t-norms. Next, we propose an algorithm for approximation of the t-norm’s generator based on the weighting method and Lawson-Hanson’s algorithm. Suitable modifications of the algorithm can generalize our solutions to aggregation operators. In this way we get new results for generated means which are well-known representatives of aggregation operators. Empirically measured data suggest that people do not understand conjunction necessarily as a commutative operation. Finally, we investigate the modelling of the conjunction via generated Choquet integral.

Practical Exhaustive Generation of Small Multiway Cuts in Sparse Graphs

We propose a new algorithm for practically feasible exhaustive generation of small multiway cuts in sparse graphs. The purpose of the algorithm is to support a complete analysis of critical combinations of road disruptions in real-world road networks. Our algorithm elaborates on a simple underlying idea from matroid
theory – that a circuit-cocircuit intersection cannot have cardinality one (here cocircuits are the generated cuts). We evaluate practical performance of the algorithm on real-world road networks, and propose algorithmic improvements based on the technique of generation by a canonical construction path.

Ali Isavudeen, Eva Dokladalova, Nicolas Ngan and Mohamed Akil, Laboratoire Informatique Gaspard Monge, Noisy-le-Grand, and Sagem Défense et Sécurité Groupe Safran, Argenteuil, France

Self-Adaptive Architecture for Multi-sensor Embedded Vision System

Architectural optimization for heterogeneous multi-sensor processing is a real technological challenge. Most of the vision systems involve only one single color sensor and they do not address the heterogeneous sensors challenge. However, more and more applications require other types of sensor in addition, such as infrared or low-light sensor, so that the vision system could face various luminosity conditions. These heterogeneous sensors could differ in the spectral band, the resolution or even the frame rate. Such sensor variety needs huge computing performance, but embedded systems have stringent area and power constraints. Reconfigurable architecture makes possible flexible computing while respecting the latter constraints. Many reconfigurable architectures for vision application have been proposed in the past. Yet, few of them propose a real dynamic adaptation capability to manage sensor heterogeneity. In this paper, a self-adaptive architecture is proposed to deal with heterogeneous sensors dynamically. This architecture supports on-the-fly sensor switch. Architecture of the system is self-adapted thanks to a system monitor and an adaptation controller. A stream header concept is used to convey sensor information to the self-adaptive architecture. The proposed architecture was implemented in Altera Cyclone V FPGA. In this implementation, adaptation of the architecture consists in Dynamic and Partial Reconfiguration of FPGA. The self-adaptive ability of the architecture has been proved with low resource overhead and an average global adaptation time of 75 ms.

Nikolajs Nahimovs and Alexander Rivosh, Faculty of Computing, University of Latvia, Riga, Latvia.

Exceptional configurations of quantum walks with Grover’s coin

We study search by quantum walks on a two-dimensional grid using the algorithm of Ambainis, Kempe and Rivosh [AKR05]. We show what the most natural coin transformation - Grover’s diffusion transformation - has a wide class of exceptional configurations of marked locations, for which the probability to find any of marked locations does not grow over time. This extends the class of known exceptional configurations; until now the only known such configuration was the “diagonal construction” by [AR08].
Filip Nalepa, Michal Batko and Pavel Zezula, Faculty of Informatics, Masaryk University, Brno, Czech Republic
Performance Analysis of Distributed Stream Processing Applications Through Colored Petri Nets

Nowadays, a lot of data are produced every second and they need to be processed immediately. Processing such unbounded streams of data is often run in a distributed environment in order to achieve high throughput. The challenge is the ability to predict the performance-related characteristics of such applications. Knowledge of these properties is essential for decisions about the amount of needed computational resources, how the computations should be spread in the distributed environment, etc. In this paper, we present performance analysis of distributed stream processing applications using Colored Petri Nets (CPNs). We extend our previously proposed model with processing strategies which are used to specify performance effects when multiple tasks are placed on the same resource. We also show a detailed conversion of the whole proposed model to the CPNs. The conversion is validated through simulations of the CPNs which are compared to real streaming applications.

Thomas Odaker, Dieter Kranzlmueller and Jens Volkert, Ludwig Maximilians Universitaet, Muenchen, Germany and Johannes Kepler University, Linz, Austria
GPU-accelerated Real-time Mesh Simplification Using Parallel Half Edge Collapses

Mesh simplification is often used to create an approximation of a model that requires less processing time. We present the results of our approach to simplification, the parallel half edge collapse. Based on the half edge collapse that replaces an edge with one of its endpoints, we have devised a simplification method that allows the execution of half edge collapses on multiple vertex pairs of a mesh in parallel, using a set of per-vertex boundaries to avoid topological inconsistencies or mesh foldovers. This approach enables us to remove up to several thousand vertices of a mesh in parallel, depending on the model and mesh topology. We have developed an implementation that allows to exploit the parallel capabilities of modern graphics processors, enabling us to compute a view-dependent simplification of triangle meshes in real-time.

Vlasta Stavova, Vashek Matyas and Kamil Malinka, Faculty of Informatics, Masaryk University, Brno, Czech Republic
The challenge of increasing safe response of antivirus software users

While antivirus software is an essential part of nearly every computer, users often ignore its warnings and they are often unable to make a safe response when interacting with antivirus software. The aim of our study was to find working connections to increase a number of mobile device users who select a premium license with more security features over a free license with a limited
level of security. We cooperated with the antivirus company ESET and more than fourteen thousand users participated in first phase of our experiment. We tested two new types of a user dialog on the Android platform. The first user dialog was designed with a text change and the other with a new button "Ask later". As a result, we found out that the text change increased the number of premium license purchases by 66% in the first phase of our experiment, the version with the "Ask later" button increased this number by 25% in the same period.

Vladimír Štill, Petr Ročkai and Jiří Barnat, Faculty of Informatics, Masaryk University, Brno, Czech Republic

Weak Memory Models as LLVM-to-LLVM Transformations

Data races are among the most difficult software bugs to discover. They arise from multiple threads accessing the same memory location, a situation which is often hard to discern from source code alone. Detection of such bugs is further complicated by individual CPUs' use of relaxed memory models. As a matter of fact, proving absence of data races is a typical task for automated formal verification. In this paper, we present a new approach for verification of multi-threaded C and C++ programs under weakened memory models (using store buffer emulation), using an unmodified model checker that assumes Sequential Consistency. In our workflow, a C or C++ program is translated into LLVM bitcode, which is then automatically extended with store buffer emulation. After this transformation, the extended LLVM bitcode is model-checked against safety and/or liveness properties with our explicit-state model checker DIVINE.
Part III

Regular Papers
Devectorization of Array Expressions for
Selective Just in Time Specialization of Neural
Ensemble Simulations

Jan Fousek
Faculty of Informatics, Botanická 68a,
60200 Brno, Czech Republic
izaak@mail.muni.cz

Abstract. The scientific software performing numerical analysis is written
either in productivity-oriented languages and environments to speed
up the development, or in efficiency-level languages to achieve high per-
formance during the execution. Selective specialization is an increasingly
popular technique allowing to translate computationally intensive parts
of the computation to lower level language with the information on both
target architecture and problem structure known at compile time. In this
paper we apply this approach to an existing software for large-scale brain
network models The Virtual Brain, whose core is written in Python. We
describe the general scheme of devectorization and translation of numer-
ical code composed of series of element-wise operations and reductions
on multidimensional arrays to C with memory optimization.

1 Introduction and Related Work

Many scientific applications are today written in MATLAB or Python, high-level
environments providing libraries and language constructs enabling the develop-
ers to quickly formulate and solve diverse problems. While they provide high
productivity and reasonable efficiency for solving medium-sized problems, per-
formance tuning is in these environments cumbersome at least. When facing the
goal of scaling an existing numerical application to larger problem sizes, it is
in some cases possible to isolate the performance-critical core and rewrite it in
some lower-level language oriented on efficiency such as C, CUDA, or OpenCL.

However it is often the case, that parts of the demanding computation change
with the changes in the problem specification, such as in the solvers for differential
equations. On the other hand the landscape of high performance architec-
tures and environments is also very diverse including multicore CPUs, several
types of accelerators (GPUs, Intel MIC), distributed and heterogeneous systems.

There are several approaches how to achieve both high productivity and
high performance: libraries of specific data structures and routines (e.g. PETSc
[1]), active libraries (e.g. OP2 [2]), or domain specific languages (e.g. Liszt [3]).
Contrary to these approaches, we aim to preserve the original implementation
in a generic high-level language. Therefore we opt for the selective specialization
(SEJITS) approach [4]. Additionally to the specialization machinery of [4], we add the parsing of multidimensional array expressions and showcase an example of memory optimization on the specializing backend.

The basic aim of our work is closely aligned with devectorization performed as part of optimization chain in other existing tools. Theano [5] compiles symbolic linear algebraical computations to low-level implementations, however it can only fuse elementwise operations on arrays of same dimensions (no reductions or broadcasts). Numexpr [6] evaluates expressions (specified as string) on NumPy arrays removing the allocation of temporary arrays for intermediate results and blocks the computation to improve data locality. However it works with simple expressions without assignments and partial rewrites of the arrays, and can not therefore fuse computations across several statements. Devectorize.jl [7] (for Julia language) performs similar optimization as Numexpr, however it parses and specializes directly the Julia code. However, it also specializes subsequent statements separately. In current C compilers loop fusion on generic codes is not a common optimization (it is not performed by GCC), and is performed only to improve data locality without optimizing memory allocation if implemented (e.g. ICC, Polly for LLVM [8]).

In this paper, we show how to adapt the Array SSA [9, 10] formalism to implicit loops implementing the operators and functions on multidimensional arrays in higher level numerical language, and use it to build a data flow DAG describing the exact data types and data dependencies. We then fuse loops implementing particular operations where possible combining and modifying two approaches: onedimensional global conflict detection and resolution [11], and multidimensional single-statement fusion validation from tensor contraction [12]. Compared to [11] the resulting approach can deal with arrays of multiple interacting dimensions, and accommodate partial array assignments. Our approach can also handle fusions across statements, while [12] is limited to single statements (only one assignment).

The work presented in this paper lays basis for building a selective specialization for solver [13] for systems of delayed differential equations used in models of neuronal ensembles. In particular we focus on translation of the functions computing the derivative for given model, which are composed of series of element-wise operations on multidimensional arrays interleaved with reduction operators such as \texttt{sum}, or \texttt{dot}. Although motivated by specialization of concrete code base, the presented approach is generic and can be directly applied to wide class of array expressions in Python with NumPy, and with some modifications also to array expressions in MATLAB.

Our aim here will be in first step to convert the vectorized code of the NumPy expressions to an intermediate representation capturing the data dependencies between the operations. In second step we show how to convert this intermediate representation to series of loop nests in C while optimizing the amount of memory used for the intermediate results.
2 Parsing and Intermediate Representation

The NumPy expressions usually contain advanced indexing constructs describing array slices and broadcasts using colon notation: $x = \text{variables}[0,:]$, where \text{variables} is $n$ dimensional array, and $x$ is $n - 1$ dimensional. These expressions can be complex, and in general it is not possible to determine the exact shapes with static analysis. Therefore we limit ourselves to only a subset of such expressions allowing for every dimension only either singular position given by constant, or using the whole dimension either explicitly by the symbol `:`.

With this limitation in place, we can derive the dimensions of all intermediate arrays and results from the dimensions of input arrays and the semantics of used operators and functions. For this we do not need the exact dimension sizes, only their number and expected equalities (parametrized array sizes, e.g. $A: (\text{dim1, dim2, dim3})$, $B: (\text{dim1, dim2})$, etc.).

The operator semantics depend on the dimensionality of the operands. The expression $a+b$ denotes an element-wise addition if at least one of the operands is an array.

2.1 Parsing and Intermediate Representation

For parsing of the Python code we use the built-in \texttt{ast} library, which converts the source code to corresponding Abstract Syntax Tree (AST). To derive the correct shapes of the intermediate values, and for subsequent optimizations we will use the Data Flow Graph (DFG) as intermediate representation (IR). We define DFG as bipartite acyclic graph $G(V, O, E, A)$ where $A$ is a set of arrays in memory, $V$ are the value nodes, and we write $v_a$ meaning that value node refers to array $a$. $O$ are the operation application nodes, and the oriented edges represent the data dependencies: $(u, v), u \in V, v \in O$ stays for value $u$ being a result of operation $v$, and $(v, u), u \in V, v \in O$ denotes $v$ being an input of operation $v$.

For scalar source codes, the DFG is easy to derive from the AST by converting it to static single assignment form (SSA)—common IR in modern compilers [14]. In SSA, every variable is allowed to be assigned only once, all additional reassignments are relabeled to create new variable. The place of assignment is refered to as a \textit{definition}. That means, that a unique symbol is created for every value a variable holds during the execution of the program and every \textit{use} of a variable points to single place of its \textit{reaching definition}. The replacement of an old value with a new one is in the SSA context refered to as a \textit{kill}. Special $\varphi$ nodes are placed at the end of blocks under control structures to merge the effect of corresponding branches.
Listing 1.1: Source code
\[
c = (a + b) \ast d \\
c = c + 4
\]

Listing 1.2: SSA form
\[
t1 = a + b \\
c1 = t1 \ast d \\
c2 = c1 + 4
\]

Fig. 1: Data Flow Graph

Constructing a DFG from the SSA is simple: every statement in SSA will correspond to one operation node \( v \in A \) with single predecessor value node \( u \in V \) representing the left hand side and multiple descending value nodes representing the arguments (or operands); see Figure 2.1 for an example. With arrays however, it is possible to overwrite only some of the values resulting in partial kills. We cannot simply invalidate the whole array, as the result is supposed to be a combination of the initial and partially overwritten values. The code in Figure 2 illustrates such case (there could be of course more complex right-hand-sides).

To correctly capture the dependencies between statements operating on parts of the arrays, we will use the Array SSA extension from Rus et al. [9,10] with some modifications. First, we expect the code to be free of any control flow expressions—this limitation will enable us to perform more optimizations during compile time, and for the class of numerical codes is not very restrictive. Second, while we allow the dimensions sizes to be symbolic (unknown at compile time), we allow explicit indexing only on dimensions with known size. Again, this limitation will enable us to perform e.g. the memory optimization during compile time, while providing the flexibility of the code in the symbolic dimensions.

Array SSA Array-SSA introduced by [9] extends the scalar SSA with new \( \delta \) nodes placed after every array definition, merging all the reaching partial definitions which still has an effect on the current contents of the array. First we need to be able to efficiently describe portions of arrays (other than by simple enumeration). For this purpose Rus et al. [9,10] introduce Uniform Set of References (USR). USR consists of linear memory access descriptors (LMAD) [15] coupled with nonlinear operations like predication operator \# and expansion \( \cup_{i=1}^{n} \). In our case, the LMAD describing \( n \)-dimensional array is a \( n \)-tuple such that access to every dimension is given by symbolical range of integers defined by offset, start, stop, and step. The cartesian product of the components of LMAD then describes symbolically a subset of the elements of a multidimensional array. For example given the array \( a \) of shape \([4,m]\), the access given by \( a[0,:) \) will be represented by \( L = \{0\} \times (0,m - 1) \) where the set for second dimension is given as an interval with start at 0 and end an \( n \). As the LMAD is a set it can be subject to set operations. USR expressions can therefore involve unions,
intersections, complements, or their combinations, and it is also possible to test
an USR for emptiness, or two USRs for equality.

In [9] the USR formalism is used in the reaching definitions analysis and con-
stant propagation. We will use it to infer correct data dependencies between the
definitions of the array parts and their subsequent uses during the construction
of DFG.

Data flow analysis  Here we show, how to use the USR information in data-
flow analysis and in code generation. Consider the code 2. The line 3 does not kill
the values defined in line 2, only the initial values of \( a \), so the lines 2 and 3 can
be executed independently (or in any order). However, both partial kills need to
be performed after the previous use on line 1 to produce correct result. On line
4, we have a use of the whole array, which depends only on the definitions on
lines 2 and 3.

We are not interested in generic code analysis, only construction of data
flow DAG – therefore we do not have to represent the \( \delta \)-nodes explicitly (for
repeated usage), but can compute the USR intersections on-the-fly and use the
information to create correct data dependency (use-def) edges, and place synchro-
nization barriers in front of every use or definition where effects of multiple
previous definitions meet.

During the construction of the df-DAG we will create nodes with new names
after every partial kill of an array. For every array \( a \) in memory, we will create
a table \( T_{a, \text{defs}} \) of currently live definitions. For a given array, the records in this
table will contain a place of the definition (value node) and USR descriptor of
the extent of the definition from this place still alive. At the beginning, the
tables \( T_{a, \text{defs}} \) contain a list of externally defined arrays (e.g. function arguments),
and are updated with every encountered definition of array \( a \) at node \( u \). If
there is not a table \( T_{a, \text{defs}} \), we create it and, add a single record containing the
 correponding value node with an USR consisting of single LMAD covering the
encountered reference to \( a \) (may not be the whole array, but only a slice). Sec-

Devectorization of Array Exp. for Selective JIT Specialization of Neural Ensamble Simul.
ond, we look into the table $T_{defs}^a$ of current reaching definitions and for every record $(v_{def}, USR_{def}) \in T_{defs}^a$, check the possible use-def pair $USR_{def} \cap USR_{use}$ for emptiness. If the intersection is not empty, than the processed reference uses values $USR_{def} \cap USR_{use}$ defined at $v_{def}$. After checking all records in $T_{defs}^a$ we collect all used definitions $v_{def}$ and create a merge barrier node $b \in A$ depending on all by edges $(b, v_{def})$. This merge node serves as a proxy in the df-DAG merging the dependencies with new value node referencing to $a$ as an output.

**Example** Let us consider the code in the listing on Figure 2, and suppose that the variable $a$ is an array of size $[2,8,3]$. The array will be described in USR as simple LMAD $L_0^a = Range(4) \times N \times Range(3)$ which will be the single entry in the corresponding table $T_{defs}^a = \{(a_0, L_0^a)\}$.

On the line 1 we have a use of whole array $a$, and definition of array $c$, both described with USRs $L_1^a = L_1^c = Range(2) \times N \times Range(3)$. On line 2 the values of $a$ are partially rewritten: the extent of the partial kill is given by USR $L_2^a = \{0\} \times N \times Range(3)$. We consult the table $T_{def}^a$ and find that $a_0$ (the input value node) is conflicting by $L_2^a \cap L_0^a \neq \emptyset$. Therefore we place a synchronization node $sync$ depending on $a_0$ data-wise. However the node $a_0$ was already used on line 1, and therefore we place a guarding logical dependence of $sync$ on $c$. We then create a new value node $a_1$ representing the synced value (for this partial kill) and use it as an input to the broadcast operation. The result of the broadcast is value node $a_2$ and we modify the table life definitions table to correctly represent the origins of values of array $a$ to $T_{defs}^a = \{(a_0, L_0^a \setminus L_2^a), (a_2, L_2^a)\}$.

Similar computation is performed when processing line 3 with another partial kill given by USR $L_3^a = \{1\} \times N \times Range(3)$. Here we however find, that there is no conflict with the partial definition on line 2, i.e. $L_2^a \cap L_0^a = \emptyset$, and therefore there is no need for synchronization with $a_3$, only with the initial values of $a_0$. Given the resulting value from line 3 is $a_4$, the live definition table will contain only two records $T_{def}^a = \{(a_2, L_2^a), (a_3, L_3^a)\}$, because all the values of $a_0$ are overwritten at this point $(L_0^a \setminus L_2^a \setminus L_3^a = \emptyset)$.

Finally on line 4 we have a use of the whole array with USR $L_5^a = L_5^a = Range(2) \times N \times Range(3)$. Comparing this to both of the records in the table $T_{def}^a$, we find, that we use the values of both of them: $L_0^a \cap L_2^a \neq \emptyset$, $L_0^a \cap L_3^a \neq \emptyset$. Therefore we create a merge node depending on the corresponding definitions $a_2$ and $a_3$. 

![Fig. 2: Partial array assignment example](image-url)
3 Memory Optimization by Loop Fusions

NumPy creates new arrays for all intermediate results, which leads to low data locality and impaired performance if the arrays do not fit into the CPU caches. The increased memory requirements can also cause problems when the arrays are large. We show how to mitigate these problems by fusing the loop nests implementing the consecutive element-wise operations, and replacing the intermediate arrays with local scalars or lower-dimensional arrays.

We already have the DFG describing the data dependencies between the arrays and the dimensions of these arrays. We will use this representation and apply the approach from [11]. First we need to detect, which arrays can be removed. This is a difficult problem, because the presence of reduction operators such as \texttt{sum} prevents us to simply fuse all the loop nests. For multidimensional arrays the reductions prevent fusing the loops corresponding to the reduction dimension. For example if the summation were on the trailing dimension, the code dimensions.

Let us first describe the approach for one-dimensional arrays and then the extension to multiple dimensions. We will follow [11] showing the approach on our DFG IR, which differs from their DFG in representing the values explicitly as nodes and allowing multi-dimensional arrays. As the subsequent analysis deals with data dependencies, we use only the \textit{data flow} edges of our DFG.

3.1 Detection of removable arrays

An array can be removed iff the access patterns are the same for all loops reading and writing to it, otherwise it is a Fusion Preventing Value (FPV). In our case, the access patterns can be twofold: element-wise operation (map) and reduction.

Consider the DFG in Figure 3. Here the function \texttt{L3} is a reduction over the whole array \texttt{a2}. Therefore, we cannot remove \texttt{a2}, which then directly prevents the fusion of \texttt{L2} and \texttt{L3} and is marked by red as FPV:

Because the data dependencies need to remain acyclic after the loop fusions, the FPV can also transitively prevent other fusions: the loops \texttt{u} and \texttt{v} can be fused iff all paths (\texttt{u,v}) in the DFG are free from any FPV. In the example above, the \texttt{L1} and \texttt{L4} operations cannot be fused, because there is a path \texttt{(L1,L4)} going over the \texttt{a4} node, which was labeled as FPV. If we removed it, the fusion \texttt{L1L2L4} would create a cyclic dependency in the graph, which cannot be resolved because it is impossible to incorporate \texttt{L3} into the fusion.

![Fig. 3: Removable arrays and fusion preventing value node.](image-url)
In order to find all removable arrays, we first go through the operation nodes \( o \in O \) and mark the inputs of all reductions as FPV. Then for all value nodes \( v \in V \) we check the transitivity condition: for given value node \( v \), we compute all the simple paths \((s, u)\) from its source \( s \) to all use nodes \( u \) using the value \( v \) (can be easily done by modified depth-first search). We check these paths for FPV and mark the value node \( v \) as removable if all these paths are FPV-free.

### 3.2 Global conflicts and dimension overlap

When removing multiple arrays, new conflicts with FPVs can arise, which previously were not captured by the transitivity condition. In following example, we can remove either array \( a_1 \) or \( a_2 \), but not both as the resulting fusion \( L_1 L_2 L_3 L_5 \) cannot include the \( L_4 \) because of the FPV \( a_3 \) and will result in cyclic data dependency.

These global conflicts can have complicated structure and its not easy in general to select the smallest set of arrays, which will be taken out of the fusion to make it valid. In [11] this is solved by 1) detecting all possible conflicts by finding elementary cycles containing FPVs, and 2) creating a linear programming problem instance describing the effects of preventing of removal of particular arrays to break the cycles.

Second type of global conflicts arises from the multidimensionality of the removed data. For example, given dependency string \( u \leftarrow a \leftarrow v \leftarrow b \leftarrow w \), if we fuse \( u \oplus v \) without node \( w \) in dimension \( i \), we cannot fuse \( v \oplus w \) without \( u \) in dimension \( j \). In other words two fusion sets can be either disjoint, or in subset relation. Therefore after removing the conflict cycles, we will check the resulting fusion candidates for legality using the approach adapted from [12].

**Conflict cycle detection** In [11] it is shown (and proofed), that every conflict lies on an elementary undirected circle satisfying two conditions:

- edges in + direction (resp. −) point to value nodes marked for removal
- at least one edge in − direction (resp. +) points to FPV.

The elementary cycles can be easily generated either by generic algorithms, or by algorithm given by [11]. The cycle presence can be for every cycle checked in one (linear) pass for both directions.

**Conflict resolution** Every conflict can be resolved by breaking the corresponding cycle. To select the smallest set of arrays removing all conflicts [11] formulate the problem with linear programming. We will adapt their formulation to our DFG by following changes:

- arrays in our DFG are represented by nodes \( v \in V \) instead of labeled edges
- we will repeat the analysis for every array dimension \( i \) independently
- we do not have to deal with multiedges as our operations have single output
at given dimension, all our arrays are of the same length which simplifies the
objective function

Fixing the dimension \( i \), for every value node \( v_a \) we add a binary variable
\( x_v \) node for breaking the corresponding cycle: remains selected for fusion and
dimension \( i \) of array \( a \) will be removed, \( x^i_v = 1 \) if the value \( v \) is not fused and
thus breaking the dependency cycle.

Then for every cycle \( \mu \) from the set of all conflict cycles \( C^i \) on dimension \( i \) we
will require at least one value \( x^i_v \) to be removed by the sum given by Equation 2.

Finally, an array dimension can be removed if all value nodes using it are
removed. This is guaranteed by Equation 3 and the variable \( x^i_a \).

\[
x_v \in \{0, 1\}, a \in A \\
\sum_{v \in \mu} x_v \geq 1, \forall \mu \in C^i \\
x^i_a \geq x_v, \forall v_a \in V \\
\min \sum_{a' \in A} x^i_{a'}
\]

**Dimension overlap** Now we have a fusion candidates marked on the DFG, who
can serve as basis for valid fusions within each dimension of the corresponding ar-
rays. However the loop nests implementing these fusions over several dimensions
cannot overlap. If such a conflict arises, it can be resolved by removing some
dimensions from fusion candidate sets, and splitting one of the conflicting fu-
sions. Again, the memory-efficient choice of the arrays for removal from a fusion
candidate sets is difficult and can be solved by linear programming as described
by Allam et al. [12]. Here we recapitulate the approach in short translating it to
the semantic of our DFG.

First for all dimension pairs \( i, j \) we need to select all pair of nodes \( (a, b) \), \( a, b \in V \) of possible conflict borders:

1. both need to be candidates for the same fusion both in \( i \) and in \( j \),
2. there is at least one fusion edge either on \( a \) or on \( b \), which is valid either in
   \( i \) or in \( j \).

![Fig. 4: Fusion preventing non-local conflict.](image-url)
Then we generate sets of value nodes on all possible paths \( P(a,b) \subseteq 2^V \) between the pairs \((a,b)\) and construct the constraints of the LP:

\[
(x^1_a + x^1_b) - (x^j_a + x^j_b) \leq 1 + \sum_{a \in P(a,b) - a,b} (1 - x^i_a) \tag{5}
\]

\[
(x^j_a + x^j_b) - (x^i_a + x^i_b) \leq 1 + \sum_{a \in P(a,b) - a,b} (1 - x^j_a) \tag{6}
\]

While the optimal objective function is not linear, [12] provide reasonable linearization in the form:

\[
\max \sum_a \sum_{i \in \text{dims}(a)} x^i_a (\text{size}(a) - r\text{size}(a,i)) \tag{7}
\]

where \( \text{size}(a) \) is a total size of array referenced by \( a \) in memory and \( r\text{size}(a,i) \) is a size of array referenced by \( a \) when the dimension \( i \) is removed by fusion. As a result, we get marked arrays and their dimensions, which are safe to remove by fusing the corresponding neighbour operation nodes.

**Fusion sets** All arrays remaining marked for removal after the conflicts resolution serve as a basis sets for fusions. We need to make sure, that all nodes on the transitive closure of the removed edges are present in same fusion. Iteratively, for every assigned array node for removal we take the source and all targets and compute all paths between these pairs. All nodes on those paths will become part of the same fusion loop—we can perform this operation safely due to the previous steps.

**3.3 Example**

Let us illustrate the previous steps from this section with an example. We will start with the df-DAG shown on Figure 5. First we mark the FPVs: input values of reduction operations. For dimension \( n \) we do not have any FPVs and we can mark all nodes for fusion in this dimension. For dimension \( m \) there is a sum reduction operator, therefore we mark the \( a \) as FPV in the \( m \) context. Then we iterate over all value nodes and check the transitive paths connecting the def and use nodes of particular value node for the presence of FPV. We find, that the other array \( b \) can be removed in the \( m \) dimension safely (\( c \) and \( e \) are inputs/outputs, and \( d \) does not have \( m \) dimension).

Next we need to detect and resolve potential global conflicts. We create all elementary cycles and find out, that the first condition is invalid on all of them as \( c \) is not marked for removal (remember, input value). Also the arrays marked for removal on dimension \( m \) are subset of the candidates for removal on dimension \( n \), so the proposed fusions are valid. Next, we will compute the fusion sets by transitively joining the operation nodes neighboring with the arrays marked for removal. The resulting groups and synthetized code is in Figure 6.
Devectorization of Array Exp. for Selective JIT Specialization of Neural Ensemble Simul.

Fig. 5: Example DFG with multidimensional arrays.

```c
float * b = (float *) malloc(n*m*sizeof(float));
for(int i=0; i<n; i++)
   for(int j=0; j<m; j++)
      b[i*m + j] = 5 * c[i*m + j];

float * a = (float *) malloc(n*m*sizeof(float));
for(int i=0; i<n; i++)
   for(int j=0; j<m; j++)
      a[i*m + j] = b[i*m + j] + c[i*m + j];

float * d = (float *) malloc(n*sizeof(float));
for(int i=0; i<n; i++)
   d[i] = sum(a[i],m);

float * e = (float *) malloc(n*m*sizeof(float));
for(int i=0; i<n; i++)
   for(int j=0; j<m; j++)
      e[i*m + j] = d[i] + c[i*m + j];
```

Fig. 6: DFG and generated code with nodes grouped by fused dimensions.

```c
float * b = (float *) malloc(n*m*sizeof(float));
for(int i=0; i<n; i++)
   for(int j=0; j<m; j++)
      b[i*m + j] = 5 * c[i*m + j];

float * a = (float *) malloc(n*m*sizeof(float));
for(int i=0; i<n; i++)
   for(int j=0; j<m; j++)
      a[i*m + j] = b[i*m + j] + c[i*m + j];

float * d = sum(a[i],m);
for(int j=0; j<m; j++)
   e[i*m + j] = d + c[i*m + j];
```
4 Conclusions and Future Work

In this paper, we have presented the steps performed during the specialization of array expressions written in NumPy to low level language while optimizing the memory required for the computation. Replacing the access to potentially large arrays of intermediate results with lower-dimensional arrays or scalars in fused operations also improves data locality for better usage of memory bandwidth and cache hierarchy.

As a next step, we will extend the devectorization into a the full specialization framework for The Virtual Brain and evaluate the performance of the specialized code. With full control over the resulting code generation, we will be able to build specific backends synthetizing for various architectures and purposes (such as parameter sweeps on GPUs).

References

Devectorization of Array Exp. for Selective JIT Specialization of Neural Ensemble Simul.

Parallel Computations of Differential Equations

Filip Kocina¹, Petr Veigend¹, Gabriela Nečasová¹, and Jiří Kunovský¹

Faculty of Information Technology, Brno University of Technology, Božetěchova 1/2, 612 66, Brno, Czech Republic
{ikocina, iveigend, inecasova, kunovsky}@fit.vutbr.cz

Abstract. The paper is focused on an original mathematical approach which uses the Modern Taylor Series Method (MTSM) for solving differential equations in a non-traditional way. Even though this method is not much preferred in the literature, experimental calculations done at the Department of Intelligent Systems of the Faculty of Information Technology of TU Brno have shown and theoretical analyses at the Department of Mathematics of the Faculty of Electrical Engineering and Communication of TU Brno have verified that the accuracy and stability of the Taylor series method exceeds the currently used algorithms for numerically solving differential equations.

1 Introduction

The Modern Taylor Series Method [5] is based on a recurrent calculation of the Taylor series terms for each time step. Thus the complicated calculation of higher order derivatives (much criticised in the literature) need not be performed but rather the value of each Taylor series term is numerically calculated [1, 2].

An important part of the method is an automatic integration order setting, i.e. using as many Taylor series terms as the defined accuracy requires. Thus it is usual that the computation uses different numbers of Taylor series terms for different steps of constant length.

An automatic transformation of the original problem is a necessary part of the Modern Taylor Series Method. The original system of differential equations is automatically transformed to a polynomial form, i.e. to a form suitable for easily calculating the Taylor series forms using recurrent formulae [9].

It has also been verified that the computation speed enabled by the newly developed Taylor series method is, while keeping the high accuracy, greater than that achieved by the algorithms currently used for numerically solving systems of differential equations. This feature is accentuated especially while solving large scale systems of linear differential equations.

The Modern Taylor Series Method also has some properties very favourable for parallel processing. Many calculation operations are independent making it possible to perform the calculations independently using separate processors of parallel computing systems.

Since the calculations of the transformed system (after the automatic transformation of the initial problem) use only the basic mathematical operations
simple specialised elementary processors can be designed for their implementation thus creating an efficient parallel computing system with a relatively simple architecture.

2 Parallel solution of linear ODEs using shared memory

We assume the system of ODEs [3, 4] in following form:

\[
\begin{align*}
y' &= k_{1:1} \cdot y + k_{1:2} \cdot z + \cdots + k_{1:m} \cdot w + constant_1 \quad y(0) = y_0 \\
z' &= k_{2:1} \cdot y + k_{2:2} \cdot z + \cdots + k_{2:m} \cdot w + constant_2 \quad z(0) = z_0 \\
&\vdots \\
w' &= k_{m:1} \cdot y + k_{m:2} \cdot z + \cdots + k_{m:m} \cdot w + constant_m \quad w(0) = w_0
\end{align*}
\]

The system of ODEs (1) consists of \( m \) equations. Each equation has:

- \( m \) terms marked as \( k_{indexOfEquation:indexOfTerm} \), which represents coefficients,
- one constant,
- one term \((y(0), z(0), \ldots, w(0))\), which represents initial condition of equations.

Differential equations from the system (1) can be rewritten as follows (see (2)). We denote the first terms of the Taylor series as \( DY_{11}, DZ_{11}, \ldots, DW_{11} \). Integration step is denoted as \( h \).

\[
\begin{align*}
DY_{11} &= h \cdot y' = h \cdot (k_{1:1} \cdot y_0 + k_{1:2} \cdot z_0 + \cdots + k_{1:m} \cdot w_0 + constant_1) \\
DZ_{11} &= h \cdot z' = h \cdot (k_{2:1} \cdot y_0 + k_{2:2} \cdot z_0 + \cdots + k_{2:m} \cdot w_0 + constant_2) \\
&\vdots \\
DW_{11} &= h \cdot w' = h \cdot (k_{m:1} \cdot y_0 + k_{m:2} \cdot z_0 + \cdots + k_{m:m} \cdot w_0 + constant_m)
\end{align*}
\]

It is possible to compute the second term of the Taylor series, denoted as \( DY_{12}, DZ_{12}, \ldots, DW_{12} \), like so.

\[
\begin{align*}
DY_{12} &= h^2 \cdot y' = h \cdot (k_{1:1} \cdot DY_{11} + k_{1:2} \cdot DZ_{11} + \cdots + k_{1:m} \cdot DW_{11}) \\
DZ_{12} &= h^2 \cdot z' = h \cdot (k_{2:1} \cdot DY_{11} + k_{2:2} \cdot DZ_{11} + \cdots + k_{2:m} \cdot DW_{11}) \\
&\vdots \\
DW_{12} &= h^2 \cdot w' = h \cdot (k_{m:1} \cdot DY_{11} + k_{m:2} \cdot DZ_{11} + \cdots + k_{m:m} \cdot DW_{11})
\end{align*}
\]

As soon as the given given precision is reached, the final result for the individual Taylor series can be calculated. During each step the Taylor series is
constructed. The result can be obtained as a sum of Taylor series terms and initial condition of the step.

\[ \begin{align*}
DY_1 &= y_0 + DY_{11} + DY_{12} + \cdots + DY_{1m} \\
DY_1 &= z_0 + DZ_{11} + DZ_{12} + \cdots + DZ_{1m} \\
\vdots \\
DW_1 &= w_0 + DW_{11} + DW_{12} + \cdots + DW_{1m}
\end{align*} \]  

(4)

2.1 Implementation

A simple implementation of Modern Taylor Series Method was written in C++ language. It uses the system of ODEs in the form (1) for computation. Some optimizations were done to reduce division operations. The only part of algorithm which can be parallelized is the calculation of individual Taylor series terms. OpenMP API [6] was used for parallelization, because it is necessary to communicate between threads and share data. Each thread performs the same (or very similar) work – static scheduling was used.

Due to the limitations of the OpenMP library, the current version uses double data type for calculation. Due to the nature of the selected experiment, this does not present a problem.

2.2 Experiments

Testing was performed on one of the available research servers at the Faculty of Information Technology, Brno University of Technology [8].

Compilation was done with an option `O3` with openMP library linked using `-fopenmp`.

Experiments were performed on the sine function for the step size \( h = 0.1 \), computation error \( \text{err} = 10^{-20} \) (sets number of Taylor series terms) and different number of steps (that changes with the end time \( t_{\text{max}} \)). Number of ODEs is 2. Results are shown in table (1). End time represents the maximum simulation time. Obtained results were truncated to 6 decimal places.

Serial solution is faster than the parallel one. This is due to the significant overhead introduced by the parallelization. There is a high amount of communication between individual threads during computation.

During each step, it is necessary for all threads to access the shared memory multiple times to update initial conditions (see equations (2), (3), (4)). For example term \( DY_{12} \) is dependent on the previous term \( DY_{11} \), and so on. This dependency is hard to overcome and it limits the potential of this approach.

3 Parallel solution of linear ODEs using distributed memory

As we could see, there is a huge overhead introduced by multi-threaded implementation in previous text. The biggest problem is communication. If we avoid communication, we get faster solution.
3.1 Adaptation of the generic formula

The generic formula of Modern Taylor Series Method for calculation of next values (5)

\[ y(t_0 + h) = \sum_{k=0}^{\text{ORD}} \frac{y^{(k)}(t_0)}{k!} h^k \]  

where \( \text{ORD} \) is maximal order, can be well adopted for parallel computations, especially for systems of linear differential equations. These systems can be described by (6).

\[ y' = A \cdot y + b \]  

where \( A \) is a matrix and \( y, b \) are vectors. This form can clearly cover up even very large systems.

Now we can express higher derivatives:

\[ y'' = A \cdot y' = A^2 \cdot y + A \cdot b \]  

\[ y^{(k)} = A^k \cdot y + A^{k-1} \cdot b \]  

and after substitution (8) into (5) we get (9).

\[ y(t_0 + h) \doteq y_0 + \sum_{k=1}^{\text{ORD}} \frac{A^k \cdot y_0 + A^{k-1} \cdot b}{k!} h^k \]  

If we split the sum, we get (10)

\[ y(t_0 + h) \doteq \left( \mathbf{I} + \sum_{k=1}^{\text{ORD}} \frac{A_k}{k!} h^k \right) \cdot y_0 + \sum_{k=1}^{\text{ORD}} \frac{A^{k-1} \cdot b}{k!} h^k \]
and generally (11)

\[
y_{k+1} = \left( \sum_{k=0}^{\text{ORD}} \frac{A_k}{k!} h^k \right) \cdot y_k + \sum_{k=1}^{\text{ORD}} \frac{A_{k-1}}{k!} \cdot b_k
\]

which can be transformed in (12)

\[
y_{k+1} = \hat{A} \cdot y_k + \hat{b}
\]

where \(\hat{A}\) is a transformed matrix and \(\hat{b}\) is a transformed vector:

\[
\hat{A} = \sum_{k=0}^{\text{ORD}} \frac{A_k}{k!} h^k
\]

\[
\hat{b} = \sum_{k=1}^{\text{ORD}} \frac{A_{k-1}}{k!} \cdot b_k
\]

The conversion into parallel version is now evident. Partial sums

\[
A_i = \sum_{k=0}^{\text{ORD}-1} \frac{h^{nk+i}}{(nk+i)!} \cdot A^{nk+i-1}
\]

are computed in \(n\) threads and there is no need for any communication or synchronization between threads except of loading the matrix at the beginning and handing back computed partial sums at the end. The reason of decreasing of exponent by one is single computation of partial sums for both transformed matrix and transformed vector (double calculation would have to be performed otherwise).

The final transformed matrix \(\hat{A}\) and the transformed vector \(\hat{b}\) are calculated afterwards:

\[
\hat{A} = \left( \sum_{i=1}^{n} A_i \right) \cdot A + I
\]

\[
\hat{b} = \left( \sum_{i=1}^{n} A_i \right) \cdot b
\]

The evaluation of function values can be done in a single thread (it’s not as demanding as previous calculations).

### 3.2 Implementation

The described transformation was written in C++ language. Object superstructure of GMP (The GNU Multiple Precision Arithmetic Library [7]) was used for numbers with high precision (double precision is not sufficient for higher orders where the results degrade by the noise). Matrices are represented by 2D arrays. Standard threads of Linux platform were chosen for parallelization. Optimization O3 was used.
3.3 Experiments

The same server was utilized and the same problem was examined as in the case of experiments with shared memory. Experiments were undertaken in 24 threads (the number of processor cores) with computation error $err = 10^{-20}$ with parameters of calculation shown in table 2. End time represents the maximum simulation time, Step size is the integration step, Order represents number of Taylor Series terms. Calculation times, Time of calc., shown in the rightmost column, are evaluated using time in Linux console. The testing software ran the calculation multiple times with given parameters, then the average time was calculated.

<table>
<thead>
<tr>
<th>End time</th>
<th>Step size</th>
<th>Nr. of steps</th>
<th>Bits/number</th>
<th>Order</th>
<th>Time of calc. [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>32</td>
<td>1</td>
<td>65</td>
<td>128</td>
<td>0.000596</td>
</tr>
<tr>
<td>64</td>
<td>64</td>
<td>1</td>
<td>65</td>
<td>193</td>
<td>0.000630</td>
</tr>
<tr>
<td>128</td>
<td>128</td>
<td>1</td>
<td>194</td>
<td>450</td>
<td>0.000847</td>
</tr>
<tr>
<td>256</td>
<td>256</td>
<td>1</td>
<td>388</td>
<td>900</td>
<td>0.001339</td>
</tr>
<tr>
<td>512</td>
<td>256</td>
<td>2</td>
<td>388</td>
<td>900</td>
<td>0.001345</td>
</tr>
<tr>
<td>1024</td>
<td>256</td>
<td>4</td>
<td>388</td>
<td>900</td>
<td>0.001349</td>
</tr>
<tr>
<td>2048</td>
<td>256</td>
<td>8</td>
<td>388</td>
<td>900</td>
<td>0.001359</td>
</tr>
<tr>
<td>4096</td>
<td>256</td>
<td>16</td>
<td>388</td>
<td>900</td>
<td>0.001407</td>
</tr>
<tr>
<td>8192</td>
<td>256</td>
<td>32</td>
<td>388</td>
<td>900</td>
<td>0.001461</td>
</tr>
<tr>
<td>16384</td>
<td>256</td>
<td>64</td>
<td>388</td>
<td>900</td>
<td>0.001570</td>
</tr>
<tr>
<td>32768</td>
<td>256</td>
<td>128</td>
<td>388</td>
<td>900</td>
<td>0.001831</td>
</tr>
<tr>
<td>65536</td>
<td>256</td>
<td>256</td>
<td>388</td>
<td>900</td>
<td>0.002259</td>
</tr>
<tr>
<td>131072</td>
<td>256</td>
<td>512</td>
<td>388</td>
<td>900</td>
<td>0.003177</td>
</tr>
<tr>
<td>262144</td>
<td>256</td>
<td>1024</td>
<td>388</td>
<td>900</td>
<td>0.005027</td>
</tr>
<tr>
<td>524288</td>
<td>256</td>
<td>2048</td>
<td>388</td>
<td>900</td>
<td>0.009455</td>
</tr>
<tr>
<td>1048576</td>
<td>256</td>
<td>4096</td>
<td>388</td>
<td>900</td>
<td>0.016428</td>
</tr>
<tr>
<td>2097152</td>
<td>256</td>
<td>8192</td>
<td>388</td>
<td>900</td>
<td>0.030459</td>
</tr>
</tbody>
</table>

Table 2. Parameters of the parallel solution

4 Comparison of the approaches

In this chapter the final comparison of serial and parallel solution which used double precision for calculations and shared memory (SM) with parallel solution using numbers with arbitrary precision and distributed memory (DM) where unnecessary communication was eliminated is performed. Table 3 shows calculation times with acceleration in the last column. Acceleration is expressed by two numbers separated by slash, where the first number is the ratio between serial SM and DM execution times and the second number is the ratio between parallel SM and DM execution times. It is evident that parallel solution with
shared memory has brought no improvement yet but we’d like to focus on decreasing communication between threads to develop parallel system for solving non-linear problems as well.

<table>
<thead>
<tr>
<th>End time</th>
<th>SM: serial</th>
<th>SM: parallel</th>
<th>DM: parallel</th>
<th>Acceleration</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>0.002803</td>
<td>0.022456</td>
<td>0.000596</td>
<td>3.36/37.68</td>
</tr>
<tr>
<td>64</td>
<td>0.002548</td>
<td>0.041203</td>
<td>0.000630</td>
<td>4.04/65.40</td>
</tr>
<tr>
<td>128</td>
<td>0.003450</td>
<td>0.077543</td>
<td>0.000847</td>
<td>4.07/91.55</td>
</tr>
<tr>
<td>256</td>
<td>0.004705</td>
<td>0.147871</td>
<td>0.001339</td>
<td>3.51/110.43</td>
</tr>
<tr>
<td>512</td>
<td>0.007243</td>
<td>0.274608</td>
<td>0.001345</td>
<td>5.39/204.17</td>
</tr>
<tr>
<td>1024</td>
<td>0.011019</td>
<td>0.560590</td>
<td>0.001349</td>
<td>8.17/415.56</td>
</tr>
<tr>
<td>2048</td>
<td>0.012770</td>
<td>1.124064</td>
<td>0.001359</td>
<td>9.40/827.13</td>
</tr>
<tr>
<td>4096</td>
<td>0.016025</td>
<td>2.282417</td>
<td>0.001407</td>
<td>11.39/1622.19</td>
</tr>
<tr>
<td>8192</td>
<td>0.029662</td>
<td>4.366353</td>
<td>0.001461</td>
<td>20.30/2988.61</td>
</tr>
<tr>
<td>16384</td>
<td>0.057105</td>
<td>8.881492</td>
<td>0.001570</td>
<td>36.37/5667.00</td>
</tr>
<tr>
<td>32768</td>
<td>0.112840</td>
<td>17.452718</td>
<td>0.001831</td>
<td>61.63/9531.80</td>
</tr>
<tr>
<td>65536</td>
<td>0.200561</td>
<td>35.351026</td>
<td>0.002259</td>
<td>88.78/15648.97</td>
</tr>
<tr>
<td>131072</td>
<td>0.372701</td>
<td>71.907891</td>
<td>0.003177</td>
<td>117.31/22633.90</td>
</tr>
<tr>
<td>262144</td>
<td>0.740928</td>
<td>140.599981</td>
<td>0.005027</td>
<td>147.39/27968.96</td>
</tr>
<tr>
<td>524288</td>
<td>1.487750</td>
<td>288.639102</td>
<td>0.009455</td>
<td>157.35/30527.67</td>
</tr>
<tr>
<td>1048576</td>
<td>2.955054</td>
<td>575.364755</td>
<td>0.016428</td>
<td>179.88/35029.51</td>
</tr>
<tr>
<td>2097152</td>
<td>5.907235</td>
<td>1125.122680</td>
<td>0.030459</td>
<td>193.94/36938.92</td>
</tr>
</tbody>
</table>

Table 3. Calculation times and acceleration

5 Conclusion

This article showed two approaches to parallelization of Modern Taylor Series Method. The former one uses shared memory and computes the solution with assistance of OpenMP using double precision of numbers while the latter one uses distributed memory and numbers with arbitrary precision.

We discussed big disadvantages of parallel solution: communication. It introduces huge overhead which makes the parallel version much slower than the serial one. Though we’d like to improve this method in further research because the other method is not suitable for solving non-linear problems.

Then parallel version using distributed memory was described. It eliminated communication to the minimum and made a choice of much larger step possible thanks to arbitrary precision of numbers. The final comparison clearly emphasized the acceleration of the latter method which introduced great acceleration.

Reviewers’ comments were accepted and reflected in the paper. Detailed information will be presented during the conference MEMICS 2015.
6 Acknowledgments

This paper has been elaborated in the framework of the project IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070, supported by Operational Programme Education for Competitiveness and co-financed by the European Social Fund and the state budget of the Czech Republic. The paper includes the solution results of the internal BUT project FIT-S-14-2486.

References

7. GMPLib.org. The GNU Multiple Precision Arithmetic Library. Available online: http://gmplib.org/
Absolutely Unlimited Deep Pushdown Automata

Jiří Kučera, Alexander Meduna, and Ondřej Soukup

Brno University of Technology, Faculty of Information Technology, IT4Innovations
Centre of Excellence, Božetěchova 1/2, 612 66 Brno, Czech Republic
ikucera@fit.vutbr.cz, meduna@fit.vutbr.cz, isoukup@fit.vutbr.cz

Abstract. This paper introduces an absolutely unlimited deep pushdown automata and studies their computational power. These automata are generalized versions of recently introduced deep pushdown automata in the terms of the depth of expansions. They can expand nonterminal pushdown symbol despite its depth. It is shown that propagating and erasing versions of absolutely unlimited deep pushdown automata characterize type 1 and type 0 languages, respectively.

Keywords: deep pushdown automata, unlimited deep pushdown automata, computational power, absolutely unlimited deep expansions

1 Introduction

Deep pushdown automata (DPDA for short) as a generalization of classical pushdown automata and the automata counterpart to $n$-limited state grammars were introduced in [3]. State grammars were introduced in [2], however, not only their $n$-limited versions. The main idea behind DPDA comes from LL syntax analysis, where the topmost symbol on the pushdown can be either popped or expanded. As it was proved in [3], allowing the expansions to be performed deeper in the pushdown will increase the accepting power of classical pushdown automata. With this feature, LL parsers are able to parse languages that are not context-free, but not every context-sensitive language, since the depth of expansions is limited. As a natural extension of DPDA, in this study we consider DPDA with no limit imposed on the depth of expansions, which are also a counterpart to unlimited state grammars.

We introduce unlimited deep pushdown automata (UDPDA for short), which stays as an automata counterpart to unlimited state grammars. An unlimited deep pushdown automaton can expand nonterminal pushdown symbols, regardless how deep in the pushdown stack they occur. We distinguish two types of determination of nonterminal to be expanded based on its position within the pushdown—absolute and relative. However, in this paper, we consider only the absolutely unlimited deep pushdown expansions, the study on the relatively unlimited deep pushdown expansions is currently being prepared.

Absolutely unlimited deep pushdown expansions are very natural generalization of $n$-limited deep pushdown expansions. The topmost nonterminal specified
Absolutely Unlimited Deep Pushdown Automata

by applied rule is expanded independently of its depth. It is proved that absolutely unlimited deep pushdown expansions provide the power of linear bounded automata for propagating versions and the power of Turing machines for erasing versions.

2 Preliminaries

We assume that the reader is familiar with formal language theory (see [4, 5]). For a set \( W \), \( \text{card}(W) \) denotes its cardinality. Let \( V \) be an alphabet (finite nonempty set). \( V^* \) is the set of all strings over \( V \). Algebraically, \( V^* \) represents the free monoid generated by \( V \) under the operation of concatenation. The unit of \( V^* \) is denoted by \( \varepsilon \). Set \( V^+ = V^* - \{ \varepsilon \} \). Algebraically, \( V^+ \) is thus the free semigroup generated by \( V \) under the operation of concatenation. For \( w \in V^* \), \( |w| \) denotes the length of \( w \). The alphabet of \( w \), denoted by \( \text{alph}(w) \), is the set of symbols appearing in \( w \).

Let \( \Rightarrow \) be a relation over \( V^* \). The transitive and transitive-reflexive closure of \( \Rightarrow \) are denoted \( \Rightarrow^* \) and \( \Rightarrow^+ \), respectively. Unless we explicitly stated otherwise, we write \( x \Rightarrow y \) instead of \( (x, y) \in \Rightarrow \).

The families of context-free, context-sensitive and recursively enumerable languages are denoted by \( \text{CF}, \text{CS} \) and \( \text{RE} \), respectively.

A state grammar (see [2]) is a 6-tuple \( G = (K, V, T, P, S, s) \), where \( K \) is a nonempty finite set of states, \( V \) is a total alphabet, \( T \subset V \) is a terminal alphabet, \( P \subseteq K \times (V - T) 	imes K 	imes V^* \) is a finite relation called the set of productions, \( S \in V - T \) is the initial symbol, and \( s \in K \) is the initial state. Instead of \( (p, A, q, x) \in P \), we write \( (p, A) \rightarrow (q, x) \in P \). Let \( \Rightarrow \) be a relation of direct derivation on \( K \times V^* \) defined as follows: \( (p, uAv) \Rightarrow (q, uxv) \) iff \( (p, A) \rightarrow (q, x) \in P \) and \( (p, A') \rightarrow \alpha \notin P \), where \( p, q \in K, A \in V - T, u, v, x \in V^*, A' \in \text{alph}(u) - T, \) and \( \alpha \in K \times V^* \). By \( (p, uAv) \Rightarrow (q, uxv) \), \( (p, A) \rightarrow (q, x) \), we express that \( (p, uAv) \) directly derives \( (q, uxv) \) according to \( (p, A) \rightarrow (q, x) \).

In the standard manner, we extend \( \Rightarrow \) to \( \Rightarrow^m \), where \( m \geq 0 \); then, based on \( \Rightarrow^m \), we define \( \Rightarrow^* \) and \( \Rightarrow^+ \). The language generated by \( G \), denoted as \( L(G) \), is defined as \( L(G) = \{ w \in T^* \mid (s, S) \Rightarrow^* (q, w), q \in K \} \). A state grammar \( G \) is called propagating, if for every production \( (p, A) \rightarrow (q, w) \in P, w \neq \varepsilon \). By \( \text{ST} \) and \( \text{pST} \), we denote the family of languages of all state grammars and the family of languages of all propagating state grammars, respectively.

Recall that \( \text{ST} = \text{RE} \) (see [1]) and \( \text{pST} = \text{CS} \) (see [2]).

3 Definitions and Examples

In this section, we define an absolutely unlimited deep pushdown automata and demonstrate them by examples.

Informally, during every computational step an absolutely unlimited deep pushdown automaton either pops or expands its pushdown. In the case, the topmost pushdown symbol is a terminal, it is compared with the current input.
symbol and if they correspond, the pushdown symbol is popped and the input symbol is read. Otherwise, the pushdown may be expanded. With an absolutely unlimited deep pushdown expansion, a nonterminal symbol is chosen and its topmost occurrence is rewritten.

The following definition is based on the original definition of deep pushdown automata, which can be found in [3].

**Definition 1.** An absolutely unlimited deep pushdown automaton (AUDPDA for short) is an 8-tuple \( M = (Q, \Sigma, \Gamma, \#, R, s, S, F) \), where \( Q \) is a finite set of states, \( \Sigma \) is an input alphabet, \( \Gamma \) is a pushdown alphabet, \( \Gamma \cap Q = \emptyset \), \( \Sigma \subset \Gamma \), \# \in \Gamma - \Sigma \) is the special symbol called bottom marker, \( R \subseteq Q \times (\Gamma - \{\#\}) \times Q \times (\Gamma - \{\#\}) \times \Sigma \times (\Gamma - \{\#\}) \times \Sigma \times (\Gamma - \{\#\}) \times \{\#\} \) is a finite relation called the set of rules, \( s \in Q \) is the initial state, \( S \in \Gamma \) is the initial pushdown symbol, and \( F \subseteq Q \) is the set of final states. Instead of \((p, A, q, x) \in R\), we write \( pA \rightarrow qx \in R \). Set \( \chi = Q \times \Sigma^* \times (\Gamma - \{\#\})^* \{\#\} \). A configuration of \( M \) is any element of \( \chi \).

Let \( a_{r}^{+} \) be a relation on \( \chi \) such that \((q, c, w, z) a_{r}^{+} (q, w, z)\), where \( q \in Q, c \in \Sigma, w \in \Sigma^*, \) and \( z \in \Gamma^* \). Then, we say that \( M \) pops its pushdown from \((q, c, w, z)\) to \((q, w, z)\). Similarly, let \( a_{r}^{-} \) be a relation on \( \chi \) such that \((p, w, uAv) a_{r}^{-} (q, w, vxv)\) iff \( pA \rightarrow qx \in R \), where \( p, q \in Q, w \in \Sigma^*, u, x, v \in \Gamma^*, A \in \Gamma - \Sigma, A \notin \text{alph}(u) \) and for every \( A' \in \text{alph}(u) - \Sigma, pA' \rightarrow q'x' \notin R \), where \( q' \in Q \) and \( \chi \) is a family of all absolutely unlimited pushdown automaton languages.

**Example 1.** Consider the absolutely unlimited deep pushdown automaton

\[
M = (Q, \{a\}, \{S, A, X, A', X', \#\}, A, \#, \{s\}, \{(f)\})
\]
with $Q = \{\langle s \rangle, \langle c \rangle, \langle 1 \rangle, \langle 2 \rangle, \langle 1' \rangle, \langle 2' \rangle, \langle f \rangle\}$ and $R$ containing rules

\[
\begin{align*}
\langle s \rangle S & \rightarrow \langle c \rangle aSAX, \\
\langle c \rangle S & \rightarrow \langle c \rangle aSA, \\
\langle 1 \rangle A & \rightarrow \langle 2 \rangle, \\
\langle 1 \rangle & \rightarrow \langle 2 \rangle, \\
\langle 1 \rangle X & \rightarrow \langle 1 \rangle X', \\
\langle 2 \rangle X & \rightarrow \langle f \rangle.
\end{align*}
\]

On $aaaa$, $M$ makes

\[
\begin{align*}
\langle s \rangle, aaaa, S\# & \stackrel{\#^0}{\rightarrow} \langle c \rangle, aaaa, aSAX\# & \stackrel{\#^1}{\rightarrow} \langle c \rangle, aaaa, aSAX\# & \langle s \rangle S \rightarrow \langle c \rangle aSAX \\
\langle c \rangle & \rightarrow \langle c \rangle aSA & \langle c \rangle S \rightarrow \langle c \rangle aSA \\
\langle 1 \rangle & \rightarrow \langle 1 \rangle X & \langle 1 \rangle X \rightarrow \langle 1 \rangle X' \\
\langle 2 \rangle & \rightarrow \langle 1 \rangle X & \langle 2 \rangle X \rightarrow \langle f \rangle \\
\langle 1' \rangle & \rightarrow \langle 1 \rangle X & \langle 2' \rangle A \rightarrow \langle 1 \rangle A \\
\langle 2' \rangle A' & \rightarrow \langle 1 \rangle A & \langle 2' \rangle X' \rightarrow \langle 1 \rangle X \\
\langle 2' \rangle X' & \rightarrow \langle f \rangle.
\end{align*}
\]

In brief, $\langle s \rangle, aaaa, S\# \stackrel{\#^n}{\rightarrow} \langle f \rangle, \varepsilon, \#$. Observe that $L(M) = L(M)_e = \{a^{2n} \mid n \geq 0\}$, which belongs to $CS - CF$.

4 Results

In this section, we prove that $AUDPD = ST = RE$ and $pAUDPD = pST = CS$.

Lemma 1. For every state grammar, $G$, there exists an absolutely unlimited deep pushdown automaton, $M$, such that $L(G) = L(M)$.

Proof. Let $G = (K, V, T, P, S, s)$ be a state grammar. Set $N = V - T$. Introduce the AUDPDA, $M = (K \cup \{f\}, T, V \cup \{\#\}, \#, R, s, S, \{f\})$, where $R$ is constructed by performing the following steps:

(i) for every $(p, A) \rightarrow (q, x) \in P$, where $p, q \in K$, $A \in N$, and $x \in V^*$, add $pA \rightarrow qx$ to $R$;
(ii) for every \( p \in K \), add \( p \# \rightarrow \overline{f} \# \) to \( R \).

**Claim 2.** Let \( (p, S) \Rightarrow_m (q, wy) \) in \( G \), where \( p, q \in K \), \( w \in T^* \), \( y \in (NT^*)^* \), and \( m \geq 0 \). Then, \( (p, w, S \#) \alpha^{-1}(q, \varepsilon, y \#) \) in \( M \).

**Proof.** This claim is proved by induction on \( m \geq 0 \).

**Basis.** Let \( m = 0 \), so \( (p, S) \Rightarrow^0 (p, S) \) in \( G \), \( w = \varepsilon \) and \( y = S \). Then,

\[
(p, \varepsilon, S \#) \alpha^{-1}(0)(p, \varepsilon, S \#)
\]

in \( M \), so the basis holds.

**Induction Hypothesis.** Assume that the claim holds for all \( 0 \leq m \leq k \), where \( k \) is a non-negative integer.

**Induction Step.** Let \( (p, S) \Rightarrow^{k+1} (q, wy) \) in \( G \), where \( p, q \in K \), \( w \in T^* \), and \( y \in (NT^*)^* \). Since \( k + 1 \geq 1 \), express \( (p, w, S \#) \alpha^{-1}(q, \varepsilon, y \#) \) in \( M \) which implies that there also exists a move \( (p, w', S \#) \alpha^{-1}(t, \hat{w}, tA\#) \) in \( M \) which follows that there must be a rule \( tA \rightarrow qx \) in \( R \). Then,

\[
(t, \hat{w}, uAr\#) \alpha^{-1}(q, \hat{w}, uAv\#) \in M
\]

and there are no other rules with the \( (t, A') \) left-hand side in \( P \), for all \( A' \in \text{alph}(u) - T \). From the first step of the construction of \( R \) follows that there must be a rule \( tA \rightarrow qx \) in \( R \). Thus,

\[
(t, \hat{w}, uAr\#) \alpha^{-1}(q, \hat{w}, uAv\#) \in M
\]

and there are no other rules with the \( tA' \) left-hand side in \( R \), for all \( A' \in \text{alph}(u) - T \). Since \( \hat{w}y = uAv \), we have \( (q, \hat{w}, \hat{w}y\#) \alpha^{-1}(0)(q, \varepsilon, y \#) \) in \( M \), which completes the induction step.

\( \square \)

By the previous claim for \( y = \varepsilon \), if \( (s, S) \Rightarrow^*(q, w) \), where \( q \in K \) and \( w \in T^* \), then \( (s, w, S \#) \alpha^{-1}(q, \varepsilon, \#) \) in \( M \). Since \( q \# \rightarrow \overline{f} \# \in R \), we also have \( (s, w, S \#) \alpha^{-1}(q, \varepsilon, \#) \) in \( M \). Thus, \( w \in L(G) \) implies \( w \in M(M) \), so \( L(G) \subseteq L(M) \).

**Claim 3.** Let \( (p, w, S \#) \Rightarrow^m (q, \varepsilon, \hat{w}y\#) \) in \( M \), where \( p, q \in K \), \( w, \hat{w} \in T^* \), \( y \in (NT^*)^* \), and \( m \geq 0 \). Then, \( (p, S) \Rightarrow^* (q, w\hat{w}y) \) in \( G \).

**Proof.** This claim is proved by induction on \( m \geq 0 \).

**Basis.** Let \( m = 0 \). Then, \( w = \hat{w} = \varepsilon, y = S \), and

\[
(p, \varepsilon, S \#) \alpha^{-1}(0)(p, \varepsilon, S \#)
\]

in \( M \). As \( (p, S) \Rightarrow^0 (p, S) \) in \( G \), the basis holds.

**Induction Hypothesis.** Assume that the claim holds for all \( 0 \leq m \leq k \), where \( k \) is a non-negative integer.

**Induction Step.** Let \( (p, w, S \#) \Rightarrow^{k+1} (q, \varepsilon, \hat{w}y\#) \) in \( M \), where \( p, q \in K \), \( w, \hat{w} \in T^* \), and \( y \in (NT^*)^* \). Since \( k + 1 \geq 1 \), we can express \( (p, w, S \#) \Rightarrow^{k+1} (q, \varepsilon, \hat{w}y\#) \) as

\[
(p, w, S \#) \alpha^{-k}(q, \varepsilon, \hat{w}y\#),
\]
where \( \chi \) is a configuration of \( M \) whose form depend on whether the last move is a popping move or an expansion.

(I) Assume that \( \chi^{p=\flat}(q, \varepsilon, \hat{w}y\#) \) in \( M \). In the greater detail, let
\[
\chi = (q, a, a\hat{w}y\#)
\]
with \( a \in T \) such that \( w = w'a \), where \( w' \in T^* \). Thus,
\[
(p, w, S\#)^{a\hat{w}}(q, a, a\hat{w}y\#)^{\hat{w}}(q, \varepsilon, \hat{w}y\#).
\]
Since \( (p, w, S\#)^{a\hat{w}}(q, a, a\hat{w}y\#) \), we have
\[
(p, w', S\#)^{a\hat{w}}(q, \varepsilon, a\hat{w}y\#).
\]
By the induction hypothesis, \( (p, S) \Rightarrow^* (q, w'a\hat{w}y) \) in \( G \). As \( w = w'a \),
\[
(p, S) \Rightarrow^* (q, w\hat{w}y) \text{ in } G.
\]

(II) Assume that \( \chi^{a\hat{w}}(q, \varepsilon, \hat{w}y\#) \) in \( M \). If this expansion is made by the rule introduced in step (ii), then \( q = f, \hat{w} = \varepsilon, y = \varepsilon \), and the induction step follows from the induction hypothesis. Therefore, suppose that this expansion is made by a rule introduced in step (i). In greater detail, suppose that \( \chi = (t, \varepsilon, uAv\#) \) and
\[
(t, \varepsilon, uAv\#)^{\hat{w}}(q, \varepsilon, u\hat{x}v\#)
\]
by using \( tA \rightarrow qx \in R \), where \( t \in K, A \in N, u \in (NT^*)^*, v, x \in V^* \), and \( \hat{w}y = u\hat{x}v \). By the induction hypothesis, \( (p, w, S\#)^{a\hat{w}}(t, \varepsilon, uAv\#) \) implies \( (p, S) \Rightarrow^* (t, wuAv) \) in \( G \). As \( tA \rightarrow qx \in R \), \( (t, A) \rightarrow (q, x) \in P \) and for every \( A' \in \text{alph}(u) - T \), there is no other rule in \( P \) with its left-hand side in the form of \( (t, A') \). Thus, \( (p, S) \Rightarrow^* (t, wuAv) \Rightarrow (q, wu\hat{x}v) \) in \( G \). Thus, \( (p, S) \Rightarrow^* (q, w\hat{w}y) \) in \( G \) since \( \hat{w}y = u\hat{x}v \).

\( \square \)

Consider the previous claim for \( \hat{w} = y = \varepsilon \) to see that
\[
(s, w, S\#)^{\hat{w}}(q, \varepsilon, \#)
\]
in \( M \) implies \( (s, S) \Rightarrow^* (q, w) \) in \( G \). Let \( w \in L(M) \). Then,
\[
(S, w, S\#)^{\hat{w}}(\hat{f}, \varepsilon, \#),
\]
which can be expressed as \( (s, w, S\#)^{\hat{w}}(q, \varepsilon, \#)^{\hat{w}}(\hat{f}, \varepsilon, \#) \). Observe that the last move is made by a rule introduced in step (ii). By the previous claim, \( (s, S) \Rightarrow^* (q, w) \), so \( w \in L(G) \). Thus, \( w \in L(M) \) implies \( w \in L(G) \), so \( L(M) \subseteq L(G) \).

As \( L(M) \subseteq L(G) \) and \( L(G) \subseteq L(M) \), \( L(G) = L(M) \). Thus, Lemma 1 holds.

\( \square \)

**Lemma 4.** For every absolutely unlimited deep pushdown automaton, \( M \), there exists a state grammar, \( G \), such that \( L(M)\{\flat\} = L(G) \), where \( \flat \) is a new symbol such that \( \flat \notin \bigcup_{x \in L(M)} \text{alph}(x) \).
Proof. Let \( M = (Q, \Sigma, \Gamma, \# ) \) be an absolutely unlimited deep push-down automaton. Set \( N = \Gamma - \Sigma \). Introduce the state grammar, \( G = (K, \Gamma \cup \{ \bar{s}, \bar{f} \}, \Sigma \cup \{ \bar{y} \}, P, \bar{s}, \bar{Z} ) \), where

\[
K = Q \cup \{ \bar{s}, \bar{f} \}
\]

and \( P \) is constructed by performing the following steps:

(i) add \((\bar{s}, \bar{Z}) \rightarrow (s, S\#)\) to \( P \);
(ii) for every \( pA \rightarrow qx \in R \), where \( p, q \in Q \), \( A \in N \), and \( x \in \Gamma^* \), add \((p, A) \rightarrow (q, x) \) to \( P \);
(iii) for every \( p \in Q \), add \((p, \#) \rightarrow (\bar{f}, \bar{y}) \) to \( P \).

**Claim 5.** Let \((p, S\#) \Rightarrow^m (q, w,y\#) \) in \( G \), where \( p, q \in Q \), \( w \in \Sigma^* \), \( y \in (\{N - \{\#}\}\Sigma^*)^* \), and \( m \geq 0 \). Then, \((p, w, S\#) \Rightarrow^*(q, \epsilon, y\#) \) in \( M \).

**Proof.** This claim is proved by induction on \( m \geq 0 \).

**Basis.** Let \( m = 0 \), so \((p, S\#) \Rightarrow^0 (p, S\#) \) in \( G \), \( w = \epsilon \) and \( y = S \). Then, \((p, \epsilon, S\#) \Rightarrow^0 (p, \epsilon, S\#) \) in \( M \), so the basis holds.

**Induction Hypothesis.** Assume that the claim holds for all \( 0 \leq m \leq k \), where \( k \) is a non-negative integer.

**Induction Step.** Let \((p, S\#) \Rightarrow^{k+1} (q, w,y\#) \) in \( G \), where \( p, q \in Q \), \( w \in \Sigma^* \), and \( y \in (\{N - \{\#}\}\Sigma^*)^* \). Observe that rules introduced in steps (i) and (iii) are not used.

Since \( k + 1 \geq 1 \), express \((p, S\#) \Rightarrow^{k+1} (q, w,y\#) \) as

\[
(p, S\#) \Rightarrow^{k} (t, w'uAv) \Rightarrow (q, w'uAv) \]

where \( t \in Q \), \( w' \in \Sigma^* \), \( u \in (\{N - \{\#}\}\Sigma^*)^* \), \( A \in N \), \( x, v \in \Gamma^* \), \((t, A) \rightarrow (q, x) \) in \( P \), and \( wy\# = w'uAv \). Express \( w' \) as \( w' = w'\tilde{w} \) with \( \tilde{w} \in \Sigma^* \), so \( \tilde{w}y\# = uAv \). By the induction hypothesis, \((p, S\#) \Rightarrow^k (t, w'uAv) \) implies

\[
(p, w', S\#) \Rightarrow^*(t, \epsilon, uAv)
\]

in \( M \), which implies \((p, w', \tilde{w}, S\#) \Rightarrow^*(t, \tilde{w}, uAv) \) in \( M \). As \((t, A) \rightarrow (q, x) \) in \( P \) and there are no other rules with the \((t, A') \) left-hand side in \( P \), for all \( A' \in \text{alph}(u) - \Sigma \), \( tA \rightarrow qx \in R \) must be the only applicable rule on \((t, \tilde{w}, uAv) \). Thus, \((t, \tilde{w}, uAv) \Rightarrow^* (q, \tilde{w}, uAv) \) in \( M \). Since \( \tilde{w}y\# = uAv \), we have

\[
(q, \tilde{w}, \tilde{w}y\#) \Rightarrow^{|\tilde{w}|} (q, \epsilon, y\#)
\]

in \( M \), which completes the induction step. \( \square \)

Consider any \( w \in L(G) \). Observe that \( w = w'\tilde{w} \), where \( w' \in \Sigma^* \). Next, observe that \( G \) generates \( w \) as

\[
(\bar{s}, \bar{Z}) \Rightarrow (s, S\#) \quad [(\bar{s}, \bar{Z}) \rightarrow (s, S\#)]
\Rightarrow^*(q, w'\#) \quad \text{(Claim 5 with } y = \epsilon)
\Rightarrow (\bar{f}, w'\#) \quad [(q, \#) \rightarrow (\bar{f}, \bar{y})]
\]

42
where \( q \in Q \), and \((s, Z) \rightarrow (s, S\#) \in P \) and \((q, \#) \rightarrow (\bar{s}, \#) \in P \) are rules introduced in steps (i) and (iii) of the construction of \( P \), respectively. Thus, \((s, Z) \rightarrow (s, S\#) \) in \( P \). Thus, \( w' \in L(G) \) implies \( w \in L(M) \), so \( L(G) \subseteq L(M) \).

Claim 6. Let \((p, w, S\#) \rightarrow^{m}(q, \varepsilon, \hat{w}y\#) \), where \( p, q \in Q \), \( w, \hat{w} \in \Sigma^* \), and \( y \in ((N - \{\#\})^*)^* \), and \( m \geq 0 \). Then, \((p, S\#) \rightarrow^*(q, w\hat{w}y\#) \) in \( G \).

Proof. This claim is proved by induction on \( m \geq 0 \).

Basis. Let \( m = 0 \). Then, \( w = \hat{w} = \varepsilon \), \( y = S \), and \((p, \varepsilon, S\#) \rightarrow^0(p, \varepsilon, S\#) \) in \( M \). As \((p, \varepsilon, S\#) \rightarrow^0(p, S\#) \) in \( G \), the basis holds.

Induction Hypothesis. Assume that the claim holds for all \( 0 \leq m \leq k \), where \( k \) is a non-negative integer.

Induction Step. Let \((p, w, S\#) \rightarrow^{k+1}(q, \varepsilon, \hat{w}y\#) \), where \( p, q \in Q \), \( w, \hat{w} \in \Sigma^* \), and \( y \in ((N - \{\#\})^*)^* \). Since \( k+1 \geq 1 \), we can express \((p, w, S\#) \rightarrow^k(q, \varepsilon, \hat{w}y\#) \) as \((p, w, S\#) \rightarrow^k(q, \varepsilon, \hat{w}y\#) \), where \( \chi \) is a configuration of \( M \) whose form depend on whether the last move is a popping move or an expansion.

(I) Assume that \( \chi \rightarrow^k(q, \varepsilon, \hat{w}y\#) \) in \( M \). In a greater detail, let

\[
\chi = (q, a, \hat{w}y\#)
\]

with \( a \in \Sigma \) such that \( w = w'a \), where \( w' \in \Sigma^* \). Thus,

\[
(p, w, S\#) \rightarrow^k(q, a, \hat{w}y\#) \rightarrow^k(q, \varepsilon, \hat{w}y\#).
\]

Since \((p, w, S\#) \rightarrow^k(q, a, \hat{w}y\#) \), we have \((p, w', S\#) \rightarrow^k(q, \varepsilon, \hat{w}y\#) \). By the induction hypothesis, \((p, S\#) \rightarrow^k(q, w'\hat{w}y\#) \) in \( G \). As \( w = w'a \), \((p, S\#) \rightarrow^k(q, w\hat{w}y\#) \) in \( G \).

(II) Assume that \( \chi \rightarrow^k(q, \varepsilon, \hat{w}y\#) \) in \( M \). In a greater detail, let \( \chi = (t, \varepsilon, uAv) \), where \( t \in Q \), \( A \in N \), \( u \in ((N - \{\#\})^*)^* \), and \( v \in \Gamma^* \). By the induction hypothesis, \((p, w, S\#) \rightarrow^k(q, \varepsilon, uAv) \) implies \((p, S\#) \rightarrow^k(t, wAv) \) in \( G \). Consider that \((t, \varepsilon, uAv) \rightarrow^k(q, \varepsilon, \hat{w}y\#) \) according to \( tA \rightarrow qx \in R \) in \( M \) with \( \hat{w}y\# = \hat{w}v \). Following the construction of \( P \), there must be a rule \((t, A) \rightarrow (q, x) \) in \( P \) introduced in step (ii). As there are no other rules with the \( tA' \) left-hand side in \( P \), there are also no other rules with the \( \hat{t}A' \) left-hand side in \( R \), for all \( A' \in \text{alph}(u) - \Sigma \). Thus, \((t, \hat{w}v) \rightarrow (q, wuAv) \) in \( G \). Finally, putting the previous sequences of derivations together, we obtain \((p, S\#) \rightarrow^k(q, w\hat{w}y\#) \) in \( G \) since \( \hat{w}y\# = \hat{w}v \).

By the previous claim for \( y = \hat{w} = \varepsilon \), if \((s, w, S\#) \rightarrow^*(q, \varepsilon, \#) \) in \( M \), where \( q \in Q \) and \( w \in \Sigma^* \), then \((s, S\#) \rightarrow^*(q, w\#) \) in \( G \). As \( P \) contains rules introduced in steps (i) and (iii), we also have \((s, Z) \rightarrow (s, S\#) \rightarrow^*(q, w\#) \rightarrow (f, w\#) \) in \( G \). Thus, \( w \in L(M) \) implies \( \hat{w} \in L(G) \), so \( L(M) \subseteq L(G) \).

As \( L(M) \subseteq L(G) \), \( L(G) \subseteq L(M) \subseteq L(G) \). Thus, Lemma 4 holds.

\[ \square \]
Theorem 7. \( \text{AUDPD} = \text{RE} \).

Proof. In [1] \( \text{ST} = \text{RE} \) was proved. From Lemma 1 and 4, \( \text{AUDPD} = \text{ST} \), which completes the proof. \( \square \)

Theorem 8. \( \text{pAUDPD} = \text{CS} \).

Proof. Following the construction of \( R \) in Lemma 1 and a proof of that lemma, it is clear that every context-sensitive language can be accepted by some propagating AUDPDA. Thus, \( \text{CS} \subseteq \text{pAUDPD} \).

Conversely, following the construction of \( P \) in Lemma 4 and its proof, we have that for every propagating AUDPDA, \( M \), there exists a propagating state grammar, \( G \), such that \( L(M)\{\♭\} = L(G) \), where \( \♭ \) is a symbol defined in Lemma 4.

Let \( \Sigma \) be an alphabet such that \( \♭ \notin \Sigma \). Since the family of context-sensitive languages is closed under linear erasing (see [4]), for every \( L \in 2^\Sigma^* \), \( L\{\♭\} \in \text{CS} \) implies \( L \in \text{CS} \). Thus, for every propagating AUDPDA \( M \), \( L(M)\{\♭\} \in \text{CS} \) implies \( L(M) \in \text{CS} \), and then \( \text{pAUDPD} \subseteq \text{CS} \). \( \square \)

Acknowledgments

This work was supported by the TAČR grant TE01020415, the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), and the BUT grant FIT-S-14-2299.

References

Improving Sketches for Similarity Search

Vladimir Mic, David Novak and Pavel Zezula
Masaryk University, Brno, Czech Republic

Abstract. Approximate similarity search techniques are getting more popular because of a high efficiency and good scalability. A key point of such techniques is a small candidate set identification, which can be done using sketches. Sketch is a compact object representation which approximates its position in a space. It consists of a bit-string. Each bit value is usually determined by the position of the object relative to some reference object(s). Its big advantages are small size and possibility of efficient comparison via Hamming distance. In this paper we formulate desirable properties of sketches, evaluate them and show the overall gain in a search quality when they are fulfilled.

Keywords: similarity search, sketch, effectiveness, efficiency, compression, big datasets, scalability

1 Introduction

Importance of an efficient query-by-example retrieval based on similarity is rising rapidly especially because of a growing amount of produced data. Moreover, wide area of use cases makes similarity search very important and challenging. Let us mention e.g. searching photos to get images containing same objects, processing video from surveillance cameras in order to detect events, e.g. crimes, or detecting specific people in area in which they are not allowed to be. Typical properties of such use cases are (1) data items cannot be absolutely ordered by relational operators $\leq$ or $<$, (2) it is desirable to retrieve several, typically dozens or hundreds most similar items [11].

One of the most common way how to express similarity of two objects, is usage of a distance function. Formally, having a domain of objects $D$, $d$ is a total function $d : D \times D \rightarrow \mathbb{R}$. The bigger the value $d(x, y)$ is, the less similar objects $x, y$ are. Natural properties of distance function $d$ are (1) non-negativity, (2) symmetry and (3) identity. Many techniques for similarity search require (4) triangle inequality property of $d(x, y)$ as well. The pair $(D, d)$ which meets these four properties is called a metric space [13]. The distance function is usually defined on the data objects describing original data items, e.g. images, plots, audio records etc. These objects are called descriptors in this paper. Each descriptor is usually represented by a vector. We use terms data object and descriptor as synonyms, since they are mapped one to one in our case.

A naive approach to retrieve the most similar objects is based on an evaluation of a distance between query object and each object in a database. Although
this approach is precise, i.e. retrieved objects are those with the minimal dis-
tance values, it is inapplicable for big datasets or for time consuming distance
functions. The descriptors may consist of high dimensional vectors, so the over-
al dataset size on a hard drive can be even several terabytes and to read all
this data is thus very time consuming. In order to overcome issues of poor scal-
ability, approximate search techniques are being developed. These techniques
do not guarantee to retrieve objects with minimal distances to the query, but
provide objects with small distances in a much lower time. Typical scenario of
approximate similarity search is (1) efficient identification of a candidate set, (2)
re-ranking of candidate set (if necessary), (3) returning set of desirable size \( k \).
Candidate set should be as small as possible, with respect to the \( k \) value, and
precise as much as possible, i.e. containing at least \( k \) objects with very small
distance to the query.

We focus on a similarity search via sketches. Sketch is an object bit-string
representation which approximates its position in a space. Each bit value is in-
dependent and somehow limits a space where the original object is. Our main
contribution is the formulation of desirable sketch properties. Furthermore, we
evaluate their contribution and analyse how difficult is to meet them. The paper
is structured as follows: Section 2 provides a related work as well as a summary
of the sketch properties which could improve a sketch approximation power. Sec-
tion 3 contains an evaluation of these properties, Section 4 and Section 5 contain
evaluation of the overall effectiveness and efficiency and Section 6 concludes the
whole paper.

2 Sketches for Fast Approximate Similarity Search

Sketch is a bit-string object which approximates a position of an original object
in a space. Sketches may be used to perform similarity search, where they are
reported to provide promising results for various data types, original vector
lengths and distance functions [4,9,12]. Typical scenario of the sketches usage
consists of a candidate set pre-selection using sketches stored in a main memory,
original objects retrieval from a hard drive, and its re-ranking using original
distance function. In order to retrieve sketches of the most similar objects, query
sketch is usually created and sketches are compared sequentially using Hamming
distance [7,9,13]. Since sketch definition is very general, there are a lot of not
satisfactorily solved problems.

**Sketch creation** The most common way to determine each bit value is the
usage of a generalized hyperplane partitioning [13] which is depicted in Fig-
ure 1. Each bit of sketch is determined using two objects, denoted as pivots. If
original object is closer to first pivot, then bit value is 0, otherwise 1. However
many techniques are being developed, such as striping technique for some metric
spaces [4], or a pure determination according threshold for a given dimension [7].
We want to investigate, which properties are important for an accuracy of sim-
Fig. 1. Generalized hyperplane partitioning and ball partitioning.

Similarity search via sketches, and possibly use them for a selection of a suitable space partitioning.

**Sketch Length** A suitable sketch length depends on many variables, such as original distance function distribution, descriptors length and original data characteristics. That is the reason why Muller-Mollina and Shinohara report as sufficient for their case the length of 64 bits [9], while Wang et al. shows a significant gain when longer sketches are used [12]. These authors propose a technique which determines a suitable sketch length from a sample set data, which we plan to run for our data in a future.

**Similarity Functions on Sketches** While traditional approaches measure sketch similarity via Hamming distance [7,9,12], Dong et al. perform an asymmetric comparison [4]. The key idea is to utilize the precise position of a query object while the most similar sketches are searched. Such approach is showed to provide higher quality results, but is more computationally demanding and less efficient.

**Efficient sketch search** For datasets of limited sizes is fully sufficient to identify candidate set using sequential scan on all sketches. In case a sufficiently accurate approximate search is performed, it is desirable to achieve a better scalability even in the phase of candidate set identification.

All these problems represent a challenging tasks to solve. Current papers focus on the narrow particular problems, but according our best knowledge, no more general comparisons are done, as well as no more theoretical papers about sketches, suitable for our case are published. E.g. Muller-Mollina et al. [9] evaluate sketches made via generalized hyperplane partitioning. They use method which should provide balanced space partitioning, however no evaluation of this
property, as well as no evaluation of the gain when this method is used is provided. Furthermore no comparison with the other space partitioning methods is done as well. Some more theoretical study of the properties of a distance function defined on sketches is provided by Flower in [5]. However in this case the Tanimoto distance is used for a sketch comparison. In none of the cited literature is the discussion about bit correlation as well as about any general sketch properties.

We focus on a desirable properties of sketches in general. The essence of using sketches for similarity search is an appropriate space transformation. Having original space \((D, d)\) (see Section 1) sketches try to approximate ordering property of function \(d\). It means, having a space of sketches \((D', d')\) where \(D'\) is a domain of bit-strings, and transformation function \(s : D \mapsto D'\), the function \(d'\) should meet implication

\[
d(a, b) \leq d(a, c) \implies d'(s(a), s(b)) \leq d'(s(a), s(c))
\]  

(1)

in as many cases as possible. While considering real-life use cases, this requirement seems to be too strong. Especially for a big datasets it is enough, when the implication holds for small values of distance \(d\), as reported e.g. by Dong et al. in [4]. It is thanks to the natural requirement on similarity search: users typically care about ordering of a few most similar objects.

Natural desired property of sketches is its small size. They contain usually dozens or several hundreds of bits [9]. Big advantage from the efficiency point of view is the possibility to calculate distance of two sketches by a Hamming distance [7,9,12,13], which we use in this paper as well.

### 2.1 Space partitioning for sketches

There are two basic space partitioning methods, which can be used to determine bit values in sketches as well. Very common approach is the usage of a generalized hyperplane partitioning (see Section 2). The other one method is a ball partitioning (see Figure 1), which determines each bit of sketch using one pivot. For each pivot is determined its fixed radius. Then bit is set to zero if and only if object distance to the pivot is lower then this radius. In this paper we use these two space partitioning methods, enhanced with our additional requirements.

Our contribution is the identification and evaluation of three requirements, which could increase an approximation power of sketches:

- balanced partitioning (space-partitioning property),
- bit clarity (distances-to-dividing-hyper-curve property) and
- independence of bits (bit-correlation property).

The first one, balanced partitioning, is adopted in many papers. It means that for each bit value exactly one half of all sketches should have set this bit to value 0 and one half to value 1. We call it \textit{space-partitioning property}. The second one relates to a bit clarity: objects should be as far from the space
We denote it as a \textit{distances-to-dividing-hyper-curve property}. Finally, it is clear that bits should be mutually uncorrelated as much as possible. We investigate absolute value of Pearson coefficient for all bit pairs. This constitutes \textit{bit-correlation property}.

Having these requirements, we analyse how difficult is to meet them and finally, how they influence the quality of similarity search in terms of Equation 1. To create a sketch, the space partitioning method must be selected firstly.

## 3 Evaluation of Space Partitioning Properties

In order to investigate a good object-to-sketch transformation, we first analyse how difficult is to meet the three properties formalized in Section 2.1, using generalized hyperplane partitioning and ball partitioning.

### 3.1 Testing data

In this paper we use \textit{DeCAF} visual descriptors as original objects from which sketches are created. DeCAF descriptor is image descriptor for visual similarity and it consists of the 4,096 dimensional vector. We select it because of its very good performance and a big dimensionality, which makes its approximation hard. Furthermore it is quite a new descriptor. It is made as an output of the last hidden layer of Krizhevsky deep convolutional neural net [6]. This neural net have been made to solve classification problems, although it is reported that the last layer of the network is a linear classifier suitable for solving recognition problems [3,10]. DeCAF descriptors are usually compared by $L_1$, $L_2$ or cosine distance. We are using $L_2$, i.e. Euclidean distance. Sets of descriptors used in this paper were gained using images from \textit{Profiset collection} \footnote{http://disa.fi.muni.cz/profiset}. The distance distribution is depicted in Figure 2.

![Figure 2](image-url)

\textbf{Fig. 2.} Distance distribution on original DeCAF descriptors.

We use three collections of DeCAF descriptors in this paper: $D_{100K}$, $D_{1M}$, $D_{3M}$ of sizes one hundred thousand, one and three million respectively. Each smaller set is the subset of the each bigger set. The smallest collection is used as a sample set. If not specified otherwise, we use set of 2,560 randomly selected
objects or its subset. Pivots are not included in any used dataset. All presented results are averaged values over 1,000 randomly selected query objects, which are not in any dataset.

### 3.2 Space-partitioning property

Considering generalized hyperplane partitioning, we use 2,560 pivots and perform partitioning using all pivot pairs, i.e. 3,275,520 couples. The goal is to select pairs dividing space into balanced parts. We define balance as relative cardinality of the smaller part. Then pivot pair meets a balance limit if the balance is greater or equal than this limit. Table 1 shows counts of pivot pairs satisfying different balance limits. Lines depict the absolute and relative counts of such pivot pairs. Since the final sketch should have length of dozens or hundreds of bits, these values are very sufficient and generalized hyperplane partitioning is very suitable, in our case, to perform balanced partitioning, when proper pivot pairs are selected. In Sections 3.3 and 3.4 we select subsets from 163,432 pivot pairs which satisfies balance limit 45%.

Considering ball partitioning, we want to (1) confirm our intuition, that for each pivot should be investigated its own radius, and (2) find out how the radii determined from the sample set divide the whole dataset. We count radii for 2,048 pivots selected in random using sample set $D_{100K}$, and then we analyse the dataset $D_{1M}$ with these radii. Balls contain 48.7% objects on average with standard deviation 0.9%. Average radius is 84.6 with standard deviation 7.7. In next experiment we investigated how many objects are in balls with fixed radius of 82.8, which was a median value. We got values from 0.0002% to 89% which shows that differences in radii are too big, and only safe approach is to analyse own radius for each pivot.

### 3.3 Distances-to-dividing-hyper-curve property

Using generalized hyperplane partitioning, we investigate how distances of objects differ from the dividing hyperplane. To count this distance we use a formula adopted from [11]. Having object $o$ and hyperplane $\eta$ determined by pivots $p_1$ and $p_2$, the object to hyperplane distance is:

$$
\text{dist}(o, \eta) = \frac{|d(o, p_1)^2 - d(o, p_2)^2|}{2 \cdot d(p_1, p_2)}
$$

This formula, based on a cosine law, can be used only with distances on metric spaces defined with a semidefinite positive property [11]. Euclidean space meets

<table>
<thead>
<tr>
<th>Balance limit</th>
<th>49.9%</th>
<th>49%</th>
<th>48%</th>
<th>45%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pivot pairs dividing space into these parts</td>
<td>3,168</td>
<td>32,610</td>
<td>65,224</td>
<td>163,432</td>
</tr>
<tr>
<td>Percentage of pivot pairs</td>
<td>0.1%</td>
<td>1.0%</td>
<td>2.0%</td>
<td>5.0%</td>
</tr>
</tbody>
</table>

Table 1. Number of pivot pairs dividing space into parts of given size.
this property [8]. Using our sample set, we count the average distance between objects and hyperplane for all 163,432 pivot pairs selected in Section 3.2. These sorted distances are depicted in first part of Figure 3. In comparison with the common distances (see distance distribution depicted in Figure 2), these distances are very small.

For ball partitioning the distance to dividing hyper-curve \( \beta \) determined by pivot \( p \) and radius \( r \) can be computed in a clear manner:

\[
\text{dist}(o, \beta) = |d(o, p) - r|
\]  

Values for 2,560 random pivots are depicted in the second part of Figure 3. It is quite surprising that distances to hyper-curve using ball partitioning are bigger than distances to hyperplane using generalized hyperplane partitioning – when is taken into account that in both cases was used partitioning dividing space into the balanced parts, in the sense of object count.

### 3.4 Bit-correlation property

Low bit correlation seems to be a very important requirement. Our aim is to solve this issue as precisely as possible, even though its complexity does not allow to identify the most uncorrelated subset as is shown in this section.

We count the bit correlation using Pearson coefficient. In more detail, let assume \( n \) sketches, each containing \( p \) bits. Then we can transform them into \( p \) inverted sketches, each consisted of \( n \) bits (see Figure 4). In this figure the sketches are represented by rows while inverted sketches are in columns. Correlation between bits in positions \( i \) and \( j \) is now equal to the correlation of inverted sketches \( i \) and \( j \). Since sketches are bit-string objects, we can slightly simplify formula for Pearson correlation coefficient into less computationally difficult form. We denote number of bits set to value 1 in a sketch \( s \) as a cardinality with notation \( \text{card}(s) \). Sketch \( u \) made as a logical multiplication of sketches \( s \) and \( t \) is denoted

\[
u = s \times t.
\]
Then having sketches $s, t$ of length $n$, Pearson correlation coefficient is equal to:

$$
\text{Pearson}(s, t) = \frac{n \cdot \text{card}(s \times t) - \text{card}(s) \cdot \text{card}(t)}{\sqrt{[n \cdot \text{card}(s) - \text{card}(s)^2] \cdot [n \cdot \text{card}(t) - \text{card}(t)^2]}}
$$

(5)

Since cardinality of each inverted sketch can be counted only once, the evaluation of all correlations is very efficient: having 2,048 inverted sketches, we count all pairwise correlations, i.e. 2,096,128 correlations in 8.2 s.

The task is now transferred into the problem of selection of $r$ least correlated inverted sketches. Having inverted sketches $s_1, \ldots, s_p$, we create matrix $M$:

$$
M[i, j] = |\text{Pearson}(s_i, s_j)|
$$

The formal definition of least correlated subset $M$ of given size is unclear since there are several possibilities how to measure the correlation of a set. We use maximum value over all pair-wise correlations, i.e.:

$$
\text{corr}(M) = \max M[i, j]
$$

(6)

since it provides the upper bound on pairwise correlations.

Selection of the least correlated inverted sketches may be viewed as a problem of searching a minimal clique of a given size $r$ in a complete undirected weighted graph with $p$ vertices. The complexity of this problem is $O(p^r r^2)$ which makes it infeasible even for our case. Therefore instead of the precise solution we run an approximate algorithm which can be classified as a sequential greedy heuristics [1] with complexity $O(r)$. Its input consists of a matrix of inverted sketches correlations $M$ and number of desired returned indexes $r$. Full algorithm with description is in Appendix. This heuristics founds a subset with the maximum correlation 0.26 when 64 inverted sketches are selected from 1,024. We don’t know the best solution, however using a brute force algorithm we proved there is no subset with the lower maximum correlation than 0.18.

Maximal pairwise correlations in returned subsets of inverted sketches are depicted in Table 2. There are results for different numbers of inverted sketches $p$ and searched subset sizes $r$ using generalized hyperplane partitioning. Second
Table 2. Maximal pairwise correlations of inverted sketches in the retrieved low correlated subsets.

<table>
<thead>
<tr>
<th>Returned set size</th>
<th>Correlations count</th>
<th>$p = 1,024$</th>
<th>$p = 2,048$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r = 32$</td>
<td>496</td>
<td>0.20</td>
<td>–</td>
</tr>
<tr>
<td>$r = 64$</td>
<td>2,016</td>
<td>0.26</td>
<td>–</td>
</tr>
<tr>
<td>$r = 256$</td>
<td>32,640</td>
<td>–</td>
<td>0.33</td>
</tr>
</tbody>
</table>

column contains number of correlations from which are selected the maximal value depicted in next columns. Thus values in second column are equal to:

$$r \cdot (r - 1) / 2.$$

For ball partitioning we are not able to get a subset with a lower correlation than 0.62 even for $p = 2,048$ and all these pivot selection strategies:

- random pivot selection,
- outlying pivots,
- covering pivots,
- cluster centroids (k-means),
- incremental selection [2].

Such result makes usage of pure ball partitioning pointless, as is discussed in next section.

4 Accuracy of Sketch Approximation

In this section, we present a quality of approximate similarity search via sketches. We perform 100-nearest neighbours queries (100-NN) and we present average results over 1,000 randomly selected query objects. Each query is evaluated using (1) sequential scan and (2) approximate search based on candidates pre-selection via sketches and its re-ranking. Candidate sets are identified using another $k$-NN queries on sketches. Results are made using 21 different candidate set sizes for each query. We use recall to express the approximation effectiveness, which in our case means to intersect the objects retrieved by sequential scan with objects retrieved by approximate search, and divide cardinality of this intersection by 100. In Figure 5 are presented results for datasets $D_{LM}$ and $D_{AM}$. Each line in a graph corresponds to sketches created with different set of pivots. Line “Div 45/55” stands for pivots dividing sample set into parts balanced to at least 45 and 55 percent (Section 3.2), “Div 45/55 Far” for these pivots with the highest average object-to-hyperplane distances (Section 3.3) and "Div 45/55 Corr" for the least correlated pivots well dividing the sample set (Section 3.4).

Whereas the fulfilling of space-partitioning property and bit-correlation property boost search effectiveness (recall) significantly – in our implementation up to 25.2% (dataset $D_{AM}$, sketch size 64 bits, candidate set size 0.42%), the distances-to-dividing-hyper-curve property brings out a problem of a very high...
Fig. 5. Effectiveness evaluation for datasets $D_{1M}$ and $D_{3M}$.

bit correlation: pivot pairs with high object-to-hyperplane distances are in our case strongly correlated, and thus the results are very poor.

We do not thoroughly evaluate sketches made via ball partitioning, since in our first tests on a dataset $D_{100K}$ its results was significantly worst, more than 5% bellow the worst curve "Div 45/55 Far". This is expectable because of a strong bit correlation (see last paragraph in Section 3.4).
5 Retrieval Efficiency Evaluation

The main reason for the usage of sketches is a gain in efficiency, i.e. the time needed for a query evaluation is reduced significantly. Moreover the approach of candidate set pre-selection and its re-ranking provides a good scalability. In this section we mention times needed for a query evaluation. For all sketches lengths, i.e. 32, 64 and 256 the processing times are very similar. Sequential scan parallelized into 8 cores, which is used for the candidate set pre-selection is very efficient. Evaluate it on 1 million sketches of length 64 bits takes only about 0.2 seconds. The overall time is thus determined mainly by original objects retrieval from a hard-drive and its re-ranking, which depends on a candidate set size linearly. Seeking, loading and re-ranking of candidate set of size 50,000 objects takes 4.0 seconds for the $D_{1M}$ and 5.1 seconds for the $D_{3M}$ respectively. For the sketches of length 256 bits, these times are bigger by 0.2 seconds. These results show, that the only limitation on sketch length is the size of main memory, since the sketches of the whole dataset should fit there. The bottleneck is the retrieval of candidate set from a hard drive.

6 Conclusions

In this paper, we focused on a technique for approximate similarity search based on sketches. We investigated three sketch properties, which were intended to improve search quality: space-partitioning property, distances-to-dividing-hypercurve property, and bit-correlation property. Using two different methods for sketch creation, generalized hyperplane and ball partitioning, we showed that pure ball partitioning method is inappropriate for a sketch creation, because of a very high bit correlations. On the other hand generalized hyperplane partitioning is very suitable. Moreover its appropriateness can be boosted significantly, since (1) it is possible to select a huge number of pivot pairs dividing space into balanced partitions, (2) from these pivot pairs it is possible to select sufficient number of those, producing low mutually correlated bit values and (3) we showed that both these properties can improve the quality of approximate similarity search even about 25% using a given size of candidate set.

As the future work we plan to investigate suitability of other space partitioning methods for sketch creation as well as pros and cons of other possible distance functions defined on sketches. Finally we plan to replace a sequential scan made on all sketches with approximate and more efficient approach as well.

Acknowledgements

This research was supported by the Czech Science Foundation project number GBP103/12/G084.
References

Appendix

Input: matrixOfCorrelations $M$, retSize $r$

$\text{minimalMaxCorr} = \text{infinity}$;
$\text{ret} = \text{emptySet}$;
replace values in a diagonal of $M$ with values 0;
replace values in $M$ with absolute values;

for $1 \ldots k$
{
    $\text{curIndexes} = \text{emptySet}$;
    add random index to $\text{curIndexes}$;
    for $1 \ldots i$
    {
        while($\text{curIndexes}.\text{size} < r$)
        {
            add most suitable index to $\text{curIndexes}$; *
        }
        $M' = \text{submatrix of} \ M \text{ determined by} \ \text{curIndexes}$;
        $\text{curMaxCorr} = \text{maximal value from} \ M'$;
        if($\text{curMaxCorr} < \text{minimalMaxCorr}$)
        {
            $\text{minimalMaxCorr} = \text{curMaxCorr}$;
            $\text{ret} = \text{curIndexes}$;
        }
        $\text{diff} = \text{getDiff}()$; **
        while($\text{in} \ M' \text{ exists value} \ M[i, j] > \text{curMaxCorr} - \text{diff}$)
        {
            if(sum of values in row $i$ is higher than in row $j$)
            {
                $\text{curIndexes}.\text{remove}(i)$;
            }
            else
            {
                $\text{curIndexes}.\text{remove}(j)$;
            }
        }
        $M' = \text{submatrix of} \ M \text{ determined by} \ \text{curIndexes}$;
    }
}
return $\text{ret}$;

Fig. 6. Heuristic algorithm for a low correlated subset selection.

The most suitable index $i$ (line marked by *) is that with minimal value:

$$z(i) = \max_{j \in \text{curIndexes}} M[i, j]$$

selected from the set $M \setminus \text{curIndexes}$. Value diff in row marked by ** is selected from interval $[0.02, 0.03]$ randomly. We use constants $k = 10$ and $i = 40,000$, even though both seems to be quite big.
**Abstract.** Next generation sequencing (NGS), besides its apparent advantages and wide adoption, also exhibits certain technical limits. In particular, the obtained data are biased with non-negligible errors. Therefore multiple NGS error correction programs were developed. We focus on hardening and improvement of one of them, Echo, which we evaluated before to be the most promising for our research. Its original code, which evolved from an early proof-of-concept prototype, was re-designed by us from scratch, with particular focus on its stability, performance, and decrease of memory footprint. We achieve speedup of almost 70 on real inputs.

The Echo algorithm is stochastic, therefore the new implementation does not yield literally comparable results. Hence we also present a statistical method of comparing the accuracy of the original and our implementations, giving evidence that the loss of accuracy is negligible.

**Keywords:** next generation sequencing, sequencing error correction, k-mer frequency, parallelization, optimization

1 Introduction

By a cruel mapping of one of the essentials of the life to the domain of computing, we can understand DNA to be just a very long sequence of *bases* — letters A, G, C, T, which are known to encode most of properties of any living being. For the purpose of this paper we dare to ignore all the extremely wide and complex

---

*Computational resources were provided by the MetaCentrum under the program LM2010005 and the CERIT-SC under the program Centre CERIT Scientific Cloud, part of the Operational Program Research and Development for Innovations, Reg. no. CZ.1.05/3.2.00/08.0144.*
biological and chemical context, and we work with this simple abstraction. Good background information can be found in e. g. [16].

In order to even start understanding the DNA role, it is necessary to be able to “read” it reliably first, which is quite a challenging technical task at the molecular level. The family of DNA-reading techniques became known as sequencing over the past decades. In particular, next generation sequencing (NGS) is experiencing rapid development as different sequencing technologies are being released to the practise [10]. Produced data differ in many features: runtime and price obtaining them, length of fragments read continuously, and error rate.

Illumina sequencing technology is still the standard for high throughput massively parallel sequencing [13]. The DNA molecules are cut, more or less randomly, into huge number of rather short fragments (approx. 100 bases) — reads, which are read independently. Although, this technology produces shorter reads [9], it has been assumed to achieve comparable results with increased genome coverage.

But sequencing is not a flawless process. Due to technical and methodological limitations, there is a certain probability of errors to be introduced. While being read, a base can be interpreted as different one or the device does not succeed reading the particular base at all. Sequencing errors then complicate assembling the short reads into longer DNA fragments, and any subsequent analysis as well. However, highly redundant coverage of a genome in NGS data enables the correction of these errors, fortunately.

Multiple NGS error correction algorithms were developed (see Sect. 4 for a brief overview) and we tested some in our previous work [4] with Echo [5] emerging to be the most suitable for our purposes in terms of quality of the results. However, this implementation suffers from pragmatic limitations — extremely long running time and big memory requirements. In this work, we are addressing these problems by an improved parallel implementation.

2 Echo Algorithm

We give only a very brief overview of the essential ideas of the algorithm here. Details on the method and its original implementation can be found in [5], the new implementation is the topic of master’s thesis [3].

2.1 The idea

The algorithm leverages redundancy in the sequencing data. In practice, the genome is covered by the sequencing data multiple times, up to 50 for usual Illumina sequencing, i. e. a particular position in the genome is reflected in 50 reads in average. The algorithm compares the reads and it searches for similarities. If certain criteria are met, the reads are said to be neighbors (i. e. they overlap). Because the error rate in the input is expected to be less than 5 % we can assume that an error that occurs in one particular read will not occur (with high probability) in most of its neighbours. Thus we can assess from neighboring
reads if a particular base is affected by a sequencing error or not. If it is, the algorithm uses the neighbours to determine the correct value.

2.2 Original implementation

There are four principal stages of the algorithm:

– **Preprocessing.** Because we cannot distinguish between the two complementary DNA strands, we have to ensure perfect symmetry in the data first. The reads are loaded one after another and their reverse complements (reading from end to beginning and substituting $A \leftrightarrow T$ and $G \leftrightarrow C$) are computed.

– **Hashing.** Mutual comparison of millions of reads for partial overlaps would not be feasible. Therefore in this stage hashes of substrings of a suitable length $k$ of the reads are computed, and indices of the reads are stored in a huge hash table. This data structure helps finding neighbours in further stages much faster\(^5\).

– **Parameter selection.** Based on sample statistics computed on the hash table, two essential parameters — minimal overlap and maximal error tolerance are computed. These determine the criteria to consider two reads to be neighbours. Simultaneously, in this stage the confusion matrix is computed. It is a 3-dimensional array ($n \times 4 \times 4$, where $n$ is a length of the read) stating the probability $p(i, X, Y)$ of finding the base $Y$ at a position $i$ in a read (i.e. in the real, error-biased sequencing data) if the matching base in the real genome is $X$.

– **Error correction.** Using the hash table and the computed minimal overlap and maximal error tolerance, neighbours of each read are determined, expected bases in the genome are computed to maximize likelihood of the reads w.r.t. the confusion matrix, and the reads are corrected accordingly.

The implementation is a set of C programs providing the individual stages linked together with a Python script. Data flow among the stages goes through temporary files, which tend to be huge, and access to them becomes a painful bottleneck. Moreover, the data structures are not optimal, e.g. using 8 bit `char` type instead of just 2 bits required to represent a base, or storing entire reads (100 bytes) where its index (4 bytes) is sufficient. Finally, due to unfortunate reuse of code some calculations are repeated unnecessarily. Also exploitation of potential parallelism is quite limited.

2.3 New implementation

On the contrary, the new implementation is a single C++ program. Hence no temporary files are used, the data flow among the stages is streamlined, and the principal bottleneck is eliminated.

\(^5\) The complexity w.r.t. the number of reads $N$ is reduced from $O(N^2)$ to $O(N)$, while $N$ may grow up to $10^9$.\[^5\]
The data structures were made as compact as possible (e.g. by using 2 bits to store a base and avoiding storage of entire reads when not necessary), yielding significant reduction in memory footprint.

With a careful analysis of the functionality the code was restructured in order to avoid redundant computations.

Finally, after rewriting the code in a compact C++ form, a straightforward parallelization of the most critical loops (using C++11 threading constructs) in all the stages was possible.

### 3 Comparison of Implementations

The Echo algorithm, as described above, is stochastic—it builds its internal data (the confusion matrix) based on samples of the input data, not the full set. Therefore, if we change the order of input processing (with code refactoring, reorganization of the hash table, parallelization, . . . ), the program yields slightly different results though it still works correctly. Consequently, we cannot compare the outputs of improved implementations with the original one literally; instead we need a more complex methodology.

#### 3.1 Overview

We aim at a sound evaluation of the implementation correctness, as well as demonstration of the new implementation speedup. However, due to enormous requirements (time and memory) of the old implementation we were not able to run full extensive tests on real-world input sizes. Therefore we approach the evaluation in three steps:

1. We take a larger ensemble (4 sets of 30 samples) of moderate size randomly generated (hence artificial, see Sect. 3.4) inputs and we run both implementations on the whole ensemble. The goal of this test is showing that either implementation gives very similar results in terms of collected statistics (Sect. 3.2) and speedup when run on multiple samples with similar characteristics, i.e. they are not sensitive to a specific input.

2. With a moderate size real-world data (M. truncatula, Sect. 3.4) we are going to show that the results are similar (both with the same statistics and speedup) to step 1. We could have replaced step 1 with such real-world data entirely, however, much larger set of different inputs would be required to make sure the algorithm is not sensitive to particular genome type and other specificities.

3. Finally, we run speedup tests with large real-world inputs (T. pratense, Sect. 3.4). This speedup is the most significant practical result of our work. Despite running extensive tests of this size would not be feasible, due to step 1 we can claim the good result is not just because of good luck with the particular input.
3.2 Collected statistics

We assess the performance and correctness of the error correction with the method of [18], though, for pragmatic reasons, we reimplement it from scratch to fit into our workflow easily. The essential statistics describing the error correction are computed by counting all bases in all reads in the input/output according to the classification:

- **True positive** (TP): an error was present and it was corrected
- **True negative** (TN): no error, untouched
- **False positive** (FP): no error but “corrected” by the algorithm to a wrong value
- **False negatives** (FN): an error was present but not corrected, or corrected in a wrong way

Given those, the overall performance of the algorithm on a particular input is characterized by its *gain* defined as

\[
\text{gain} = \frac{TP - FP}{TP + FN}
\]

The number expresses percentage of corrected errors out of the total, decreased by the percentage of wrongly introduced errors (the FPs).

3.3 Evaluation workflow

The entire workflow is shown in Fig. 1. The optional first stage, the *genome generator* produces a sufficiently long artificial “genome”, a sequence of letters A, C, G, T with uniform random distribution.

Confusion matrices are also generated in advance according to the required error rate by *matrix generator*.

Then the *sequence generator* is run. It chooses a random position in the genome and records it to a *position map* working file (to be used to identify the read when counting statistics later). 100 bases of the simulated read are extracted, and eventual errors are introduced to them according to the probability given by the input confusion matrix. The process is repeated until the required coverage is achieved.

The generated reads are fed as an input to the tested NGS error correction program. Its output, the corrected reads, are analyzed by the *quality measurer*, using the position map recorded by *sequence generator*. Hence we know, for each read in the dataset, what it should have been without errors, and which errors were introduced to it. Therefore the statistics can be computed reliably.

3.4 Testing data sets

The core of the tests was done on randomly generated data, five sequences of 1,000,000 bases with uniform distribution of A, G, C, T. This is a very rough
approximation of a genome, where the occurrence of the letters is not uniform typically, and where many repeated patterns are present. However, with respect to the Echo algorithm, the uniform random input represents the worst case — repeated patterns etc. allow the algorithm to detect more matches, yielding more accurate outputs. Moreover, this tests are also neutral w.r.t. specificities of genome types, different species etc.

The size is sufficient for comparing accuracy of the implementations — for the purpose of reasonable efficiency, when fixing an error on a particular position, the algorithm does not scan all the matching reads but their small random sample only. With this input size, the process is already saturated, there are enough matching reads to choose from.

We also generated $2 \times 6$ 3D confusion matrices. The first 6 have 1% average probability of changing a base which is the usual rate appearing in a real sequencing process. The other 6 have 5% such probability which is a safe upper bound (up to 2.5% for Illumina is reported in [6]). Higher rate of differences would shift us from NGS errors to genome polymorphism (differences between individuals) which is not the subject of this work.
Therefore, using the sequence generator described above we were able to
generate, for a given error rate 1% or 5% and chosen genome coverage, a set of
30 input datasets (5 random “genomes” times 6 confusion matrices).

Further, in order to check for eventual differences for random vs. real data
we take a fragment of similar size (883,560 bp) of chromosome #2 of Medicago
truncatula Gaertn. [19] we used in our previous work [4]. We treat it in the same
way as the random “genome” otherwise.

Finally, we evaluate speed of the algorithm with de novo real NGS data on
Trifolium pratense L. [4], approx. 418 Mbp; a reference genome was not known
in the time of preparation of this article, therefore accuracy of the error correction
could not be evaluated directly. The dataset has 244 millions of reads of length
92, yielding approximate coverage of 54.

### 3.5 Results: accuracy

The essential quantity which describes the overall behaviour of any sequencing
error correction algorithm is the gain defined in Sect. 3.2. Table 1 shows the gain
achieved by both the implementations on the randomly generated inputs.

<table>
<thead>
<tr>
<th>Coverage</th>
<th>Error</th>
<th>Old gain</th>
<th>New gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1%</td>
<td>0.876 ± 0.004</td>
<td>0.846 ± 0.004</td>
</tr>
<tr>
<td>10</td>
<td>5%</td>
<td>0.751 ± 0.012</td>
<td>0.718 ± 0.009</td>
</tr>
<tr>
<td>50</td>
<td>1%</td>
<td>0.909 ± 0.008</td>
<td>0.910 ± 0.008</td>
</tr>
<tr>
<td>50</td>
<td>5%</td>
<td>0.904 ± 0.006</td>
<td>0.896 ± 0.003</td>
</tr>
</tbody>
</table>

The cases with coverage of 10 serve mostly as benchmarking of our evaluation
method. According to expectation, we see an apparent drop while going to 5%
error rate, probably reaching limits of the algorithm itself. In these cases the
performance of the new implementation is slightly worse.

On the other hand, coverage 50 is realistic, as well as the range 1%-5% of
error rate. In these cases, the implementations are not distinguishable in terms of
statistics, which is a great result — there is no visible trade-off for the significant
speedup (see below).

We analyze the results more deeply by looking into the structure of the
gain formula. The principal contribution comes from false negatives, i.e. cases
when the algorithm failed to correct an existing error. Table 2 summarizes their
counts. Rows with different coverage must be compared carefully — because the
input dataset is larger, the total number of errors is larger too, hence the higher

---

number of FNs. After normalization (i.e. dividing by coverage), we get expected results, increasing the coverage $5 \times$ reduces FNs $1.4 \times$ (at 1% error rate) and $2.6 \times$ (5% error rate). Again, with the realistic coverage of 50, the implementations are indistinguishable.

Table 2. Counting false negatives and positives (not normalized wrt. coverage). FPs are quite rare, usual statistics are not useful; instead the notation $m \times n$ means “in $m$ datasets exactly $n$ FPs occurred”

<table>
<thead>
<tr>
<th>Coverage</th>
<th>Error</th>
<th>FN old</th>
<th>FN new</th>
<th>FP old</th>
<th>FP new</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1%</td>
<td>12100 ± 600</td>
<td>15000 ± 600</td>
<td>0</td>
<td>2 \times 1</td>
</tr>
<tr>
<td>10</td>
<td>5%</td>
<td>123000 ± 8000</td>
<td>139000 ± 8000</td>
<td>342 ± 29</td>
<td>526 ± 26</td>
</tr>
<tr>
<td>50</td>
<td>1%</td>
<td>44000 ± 4000</td>
<td>44000 ± 4000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>5%</td>
<td>238000 ± 18000</td>
<td>259000 ± 12000</td>
<td>2 \times 1 \times 3 \times 2 + 9 \times 1</td>
<td></td>
</tr>
</tbody>
</table>

Finally, we look at false positives, i.e. when the algorithm introduces an error where it was not present. The results are given again in Tab. 2. In the 10-5% case the number rises, witnessing the limits of the method, apparently there is not enough information to produce reliable result. And this is the only criterion where the new implementation is worse in the 50-5% case. Unlike the old one, which introduce only a single FP in only 2 of 30 testing datasets, the new one brings 15 FPs altogether in 12 of the same 30 datasets. However, as there are no FPs in the 50-1% case, we do not consider it to be critical.

A quick check with a real genomic data (M. truncatula) and the fast 10-1% setup yields similar results: gain of 0.88 (old) and 0.84 (new), 10000 and 13000 false negatives (matching the smaller genome of 883,560 bp), and just 1 false positive with the new implementation. This shows that the implementations behave similarly on both randomly generated and real genomic data, confirming feasibility of the evaluation on the random data sets.

3.6 Results: speedup on test data

Speedup measurement was done on a dedicated machine (iMac, i7@3.4 GHz, 32 GB RAM, OS X 10.10.3, using Apple LLVM 6.1.0 C++ compiler). Table 3 shows running times of the original and new implementation run on the randomly generated inputs, both were run on 8 available cores.

All the computations tend to be slower for the 1% error rate (even in the 10-1% case, despite it’s hidden in the statistics, it is true for 21 of the 30 datasets). We assume that the lower error rate yields more matches of the hash values, resulting in higher congestion of the shared data structures. The phenomenon is less apparent with the original implementation because of being masked by its other inefficiencies. Similarly, the higher coverage brings more congestion therefore the running times are, in general, longer than $5 \times$, which would correspond to the input data size.
Table 3. Program running time on the randomly generated inputs. Averages and standard deviations over 30 samples of input datasets in each row.

<table>
<thead>
<tr>
<th>Coverage</th>
<th>Error</th>
<th>Old time (s)</th>
<th>New time (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1%</td>
<td>246 ± 13</td>
<td>36.7 ± 1.4</td>
<td>6.7 ± 0.5</td>
</tr>
<tr>
<td>10</td>
<td>5%</td>
<td>241 ± 6</td>
<td>27.4 ± 0.8</td>
<td>8.8 ± 0.3</td>
</tr>
<tr>
<td>50</td>
<td>1%</td>
<td>1440 ± 70</td>
<td>297 ± 10</td>
<td>4.86 ± 0.25</td>
</tr>
<tr>
<td>50</td>
<td>5%</td>
<td>1006 ± 15</td>
<td>144 ± 4</td>
<td>6.97 ± 0.14</td>
</tr>
</tbody>
</table>

The main result of this experiment is that the algorithms behave consistently (differences in the running time are low) on the fairly high number (30 samples) of different input datasets having the same characteristics. Therefore we can expect similarly consistent running times also on much larger inputs, which we were not able to test in so many different instances due to limited access to computing resources.

Running times on the *M. truncatula* data are similar, as expected.

3.7 Results: speedup in the real-world scenario

The achieved speedup of the new implementation over the old one — 5–9× with the moderate-size testing data sets is satisfactory but not groundbreaking.

The principal result is the speedup on the real input — the NGS data of *T. pratense*. The original implementation runs 40 days\(^7\) and require more than 1 TB RAM. The new implementation reduces the time below 14 hours, yielding speedup of almost 70, and the memory footprint down to 422 GB.

The higher speedup can be explained with two factors. First, the overhead of disk operations and non-optimality of data structures in the original implementation becomes more significant with growing input size. Second, the size of the critical hash table in the original implementation is fixed, while it is adaptive in the new implementation. Therefore the number of hash collisions is reduced, yielding better overall performance.

4 Related Work

The topic of NGS error correction is well covered in literature, the survey [18] provides a good overview. So far, three major types of error-correction methods were published: *k*-spectrum based, suffix tree/array-based and MSA-based methods (MSA stands for multiple sequence alignment).

The *k*-spectrum-based methods decompose the reads into a set of all *k*-mers present in them and this way *k*-spectrum is created [1]. *k*-mers within a small Hamming distance from each other are likely to belong to the same genomic

\(^7\) Because of the memory requirements the tests were run on HP ProLiant DL980 with E7-2860@2.26 GHz hence the absolute running times are not directly comparable. However, the relative speedup is the result of interest.
location. The alignment is achieved by identifying such a k-mer set. The error correction can then be applied by converting each constituent k-mer to the consensus. This method is introduced in [1, 12] and implemented with certain modifications in SOAPdenovo assembler [8], and the error correctors Quake [6], Reptile [17] and Hammer [11].

Suffix tree/array-based method is a generalization of the k-mer-based approach. Specifically, it uses multiple k values and the corresponding threshold M. Software tools SHREC [15] and HiTEC [2] implement these approaches.

MSA based method identifies reads colocalized on the unknown reference genome by using k-mers as seeds. First, a hash table is created recording all the reads to which each k-mer belongs. Second, each read r is taken as the reference, then any read r′ that shares at least one k-mer with r is identified and aligned with r using Needleman–Wunsch algorithm. A consensus sequence is created after alignment, which is taken as the new reference to be aligned with the remaining reads. A software tool Coral [14] follows this approach. An extension and improvement of this algorithm is implemented in software Echo [5] we focused on.

We have more extensive direct experience with Reptile, Quake, and Coral. In our previous evaluation, Reptile was apparently less sensitive to sequencing errors than Echo. We were not able to run Quake on the entire T. pratense dataset, the program terminates with an error indicating overflow of internal data structures. Coral, which uses slightly different approach to multiple alignment, is, besides substitution errors, also able to handle insertion errors. However, according to our preliminary experiments during preparation of [4] it yields significantly higher numbers of false positives.

Zhang et al. [20] take a similar approach to generate test data. Their script starts with an input genome, and it splits it into randomly generated reads to achieve given coverage, exactly in the same way as we do. However, the script is much simpler in terms of perturbing the generated reads with random error. We control this process more tightly with the pre-generated confusion matrices.

The survey [18] uses similar approach to assess quality of error correction, with a known reference genome in the background, same statistics of true/false positives/negatives, and definition of the gain. We did not use their implementation for pragmatic reasons — they work with real sequencing data, therefore prior alignment of the reads is required, making the toolkit dependent on other software BWA [7], which brings in dependencies on obsolete and incompatible Python packages in turn. Therefore we chose to reimplement the method, which is very simple otherwise, from scratch.

5 Conclusions

We re-engineered the implementation of the Echo NGS error correction algorithm by redesigning the original code, based on proof-of-concept prototype. The new

We use these also to assess accuracy of their reconstruction in intermediate steps of the algorithm. However, this evaluation goes beyond the scope of this paper.
implementation, besides being more compact, streamlined, homogeneous and parallel C++ code, focuses on performance and memory efficiency. We achieve a good speedup of 5–9 with moderate-size input data. With real-world large input data the speedup is almost 70. Memory footprint is decreased by more than half. In absolute numbers it means reducing computing time from weeks or months to hours or days, and memory requirements from more than 1 TB below 500 GB. This is a strong qualitative shift in the work of the researcher, allowing him/her to process genomic data of organisms of interest for the state-of-the-art research routinely.

The gain is not entirely for free, the new implementation tends to introduce slightly more false positive errors. However, these numbers are not critical as long as certain (realistic) quality of the input data (genome coverage and sequencing error rate) is preserved. In this work we demonstrate, by thorough computational experiments in a controlled environment, that this is the only impact on the accuracy of the new implementation; in all other criteria it yields similar results as the original one.

In future we will attempt to analyze the reasons for increased rate of false positives deeply, trying to eliminate the problem. We also plan to work on specialized high-throughput lookup data structures which will let us convert the implementation further to a massively parallel version.

References

Iterative Optimization of a Multi-compartmental Air Quality Modelling System

Martin Tomáš¹, Tomáš Rebok² and Christos Efstathiou³

¹ Faculty of Informatics, Masaryk University, Brno, Czech Republic,
² Institute of Computer Science, Masaryk University, Brno, Czech Republic,
³ Research Centre for Toxic Compounds in the Environment (RECETOX), Masaryk University, Brno, Czech Republic

Abstract. In this paper, we present an application for iterative optimization of a multi-compartment environmental fate and transport modelling system based on atmospheric measurements of Persistent Organic Pollutants (POPs) from the European Monitoring and Evaluation Programme (EMEP). The modelling framework involved linking science models such as the Weather Research and Forecasting (WRF) model and an experimental version of the Community Multiscale Air Quality System (CMAQ) that includes treatment of POPs species in the atmosphere, and an soil compartment that simulates the soil-air exchange. The initialization step of multi-compartment models for POPs is plagued by uncertainties in estimating the current soil burden and spatial distributions. The goal of this work is to demonstrate a first application of the modelling framework which aims to improve the reliability of modelled POP estimates in air by controlling the additional fluxes from the soil compartment. Using several machine learning methods, the system is able to make the current simulations better correspond to real complex processes occurring in the environment.

1 Introduction

The importance of cyclic transport of persistent organic pollutants (POPs) between media in the environment has been addressed in recent scientific literature. To study these transports, a multi-compartment modelling system consisted of a mesoscale meteorological model (Weather Research and Forecast model – WRF) and an enhanced air quality system (Community Multiscale Air Quality System – CMAQ⁴) was employed to simulate the fate and transport of POPs in Europe. However, it has been observed that the estimation of the initial burden of the chemical within the additional compartments (e.g. soil) is plagued by large uncertainties, affecting the quality of the model output. Measurements of POPs in their respective compartments (i.e. air, soil, water) are difficult and expensive to obtain, and establishing a measurement network such as EMEP is a complex effort.

⁴ https://www.cmascenter.org/cmaq/
Even though output from global models for historical simulations provides with the best estimates for the initial soil burden, the coarse spatial resolution makes its usage in regional models difficult while at the same time exposes the large uncertainties. The goal of this collaborative effort is to develop a system, which is able to improve the reliability of initial POPs concentrations in the soil, so that the simulations run in the virtual environment better correspond to real measurements performed in the area of interest, allowing more reliable results and better understanding of the modelling system and its components. The overall idea is to iteratively run a set of simulations, evaluate the model output against real observations at the end of every iteration, and develop a method that accounts for the information from the observations resulting in improved inputs for the compartment of interest.

These are not precise enough and thus affect the simulation results which behave differently than in the real environment. Being able to make these simulations better correspond to real measurements, one would be allowed to study the impacts of these pollutants not only on the environment itself, but on the health of the human population as well.

Fig. 1: Initial soil concentrations of BaP as available from the global model.

2 Proposed application overview

During our work, a fully automated application employing the multi-compartment version of CMAQ was developed. It runs a set of iterations, each of which aiming to make the soil POPs concentration guess more precise, so that the CMAQ simulation results better correspond to the provided air pollutants measurements.
Once the simulation results are close enough to the observed air measurements – i.e., the mean error is below a pre-defined threshold or a maximal number of iterations has been reached – the application finishes, providing the POPs concentration guess together with several quality indicators to its output.

The tasks performed during every iteration are depicted in the Figure 2. At the start of every iteration, the input data for the CMAQ simulations performed in the particular iteration have to be prepared. These input data are based on predictions made with the learning module during previous iteration. Special case of data preparation is the first iteration which cannot use the learning module – here, the inputs are generated from initial soil concentrations with semi-random methods (see section 4.2).

During input data preparation, the volatilization process takes place – meteorological and grid data together with our predicted inputs are used for calculation of volatilization (emission of the particular pollutant into air). Its values are then saved (inserted) to emission files used for CMAQ simulations, which are run subsequently.

Once these simulations finish, their output data are processed so that the soil POPs concentration guess for the subsequent iteration could use these results to make the subsequent simulation results closer to the desired air concentrations. This is the most challenging part of the overall process, since the POPs concentration guesses used for the parallel calculations have to be made sophisticatedly. The reason is that just a limited number of CMAQ runs during every iteration is possible; otherwise the whole process would take unacceptably long time.

Since there is an obvious trade-off between the convergence speed and the number of simulations run in every iteration, sufficient computing resources are highly welcomed (however, not necessary). Thus, the application is ready to use resources provided by high-performance computing centres, usually available through a job scheduling system. During our development and tests, we use computing and storage resources provided by the Czech national grid infrastructure MetaCentrum\(^5\) and by the CERIT-SC computing centre\(^6\), where the

\(^5\) http://metavo.metacentrum.cz/
\(^6\) http://www.cerit-sc.cz

---

**Fig. 2:** The set of operations taking place during every iteration.
individual simulations are submitted as embarrassingly parallel jobs and evaluated after every iteration. This allows us to tune the prediction methods in reasonable time, focusing primarily on the prediction quality rather than the convergence speed.

3 Architecture overview

The application employs modular approach, i.e., it consists from a set of modules, each of which focusing on a defined task. This approach not only makes the functional tests easier, but especially simplifies future modifications (e.g., utilization of new machine learning methods) since just a defined module has to be updated/replaced when necessary. All the modules could be divided into the following categories:

- Initial modules – these modules take care of analysis of the area, split cells to regions and prepare SQL database.
- Data preparation – after every CMAQ simulation, data are automatically grouped together. Modules in this category prepare data for the learning module as well.
- Learning module – during every iteration, the learning process is run. This process extremely helps with new inputs preparation for the next iteration.
- Volatilization module – from concentrations of POPs in the soil, appropriate air concentrations have to be computed. These POP concentrations in the air are further used as inputs for next CMAQ simulation.

As depicted before, the overall goal is to apply several learning methods able to improve the initial soil POPs concentrations input – for example, taken from the global model – and make it more precise so that the simulations employing these concentrations better correspond to real measurements performed within the area of interest.

4 Initial modules

Initial modules are run just once – during the first iteration. Their task is to prepare all the supporting files that will be needed later by learning module as well as the other ones. SQL database is also initialized during this step. A few operations accomplished by initial modules which deserve special attention are described in the following sections.

4.1 Area of interest analysis and regions identification

An analysis of the chosen area is done in the beginning of the first iteration. This analysis obtains all important properties about every cell of the area matrix (either from CMAQ input files or from external files provided by user of the application) – examples of such properties are temperature, soil type or sum of
organic carbon in the soil (list of all properties and their importance is more deeply discussed in [1]).

Since concentrations of BaP are often related to other known information (such as organic carbon concentration in the soil, temperature, etc.), some learning methods introduced later use regions (a group of one or more cells with the same properties) instead of independent area cells. Thus, the module also identifies such regions in the input area matrix; these region cells are further processed uniformly, i.e., all the operations performed to a single cell of a region apply for all the other cells in the same region as well.

4.2 Input generation for the first iteration

The first iteration is a special one, because the learning module cannot be used to generate new soil POPs concentrations guess – there is no information from the previous iterations that could be used for learning, therefore some other methods have to be used to cope with this problem.

To generate inputs for the first iteration, sufficient randomness on the input POPs soil concentrations provided to the application is used: at first, a random number of cells is selected and an uniform distribution is used to slightly change the values of these selected cells. To compute the values of the other (unselected) cells, three possible approaches are used:

1. interpolation of missing values – values of the unselected cells are interpolated from the selected cells,
2. linear model – linear function is trained with selected cells and their properties. Unselected cells values are then predicted based on their properties,
3. non-linear model – instead of linear, non-linear function is used.

Fig. 3: Examples of inputs generated during the first iteration.
Examples of inputs generated with these methods can be seen in the Figures 3a and 3b. Unlike the first one, the second one uses cell properties for prediction of values of unselected cells.

5 Volatilization module

The processes involved in the module for the soil-air exchange, include deposition, volatilization, and soil degradation terms which can be described by the following equation:

\[
\frac{\partial C_{soil}}{\partial t} = F_{dep} - F_{vol} - k_{soil} \cdot C_{soil}
\]  

(1)

As outlined by previous studies [2, 3], the flux for a surface applied chemical can be predicted using the following equations:

\[
F_{vol} = C_{air} \cdot \exp(\pi t) \cdot \sqrt{\frac{D}{\pi t}} \cdot (1 - \exp(-\frac{D^2}{4 \pi d_{air}}))
\]  

(2)

A critical parameter is the initial soil concentrations $C_{soil}$ that is obtained from global scale model simulations. The CMAQ model will calculate atmospheric concentrations at the surface layer based on estimated $F_{vol}$. These gridded concentrations can be correlated to atmospheric measurements from the MONET-EU network. This way the initial value for $C_{soil}$ can be adjusted, if unreasonable.

Volatilization over water area isn’t implemented yet, therefore learning module isn’t used for prediction of cell values in these areas.

6 Variability among predicted inputs

Unfortunately, CMAQ calculations are very time consuming and can effectively use just a very limited number of CPUs (for more details about CMAQ performance see [4]) – increasing number of available CPUs above this limit doesn’t improve the calculation speed at all. On the other hand, it is necessary to perform enough iterations/computations in order to make the learning modules able to simulate the complex CMAQ processes best. Thus, to overcome this trade-off and create enough samples for the learning modules, several CMAQ simulations are run during every iteration, benefiting from the computing power available from the computing centres used.

To allow this, learning module have to prepare several different inputs to make sure that CMAQ simulations have all the needed data for the following iteration. Some kind of variability among inputs have to be provided, so every simulation helps the application to learn something new and tries different inputs’ options.
To provide this variability, the application computes geographical distance or correlation between input cells and all the provided measurements. Based on this data, measurements are ordered and divided into several groups. The first group contains measurements that are closest or have the highest correlation to predicted cell (or region). The last group contains measurements that are furthest or have the smallest correlation. User of the application can define breakpoints which describe how the measurements are divided into groups. The number of such groups then corresponds to the number of CMAQ simulations performed during the next iteration.

Every input then uses different number of groups of measurements during prediction. First version of input is predicted only from the closest measurements or from ones with the highest correlation, therefore it uses only first group of measurements. Second version of input uses first two groups of measurements. Last version of input uses all the measurements (all groups) during prediction.

7 Learning module

The learning module is the most complex part of the designed application. This module is launched only once per an iteration and prepares new inputs for the following iteration. The main building blocks of the learning module are shown in the Figure 4.

The learning module contains an adapter of the machine learning method and other connectors for pre and post processes and learning samples creations. These links connect the core of the learning module with learning method scripts that perform specialized operations. In contrast to the core, changes in these scripts are expected in the future. That is why operations performed with them are dissected from the core. Adapter of the machine learning method enables a user to select among different artificial methods without any changes to the rest of the application.
Regarding the learning process itself, the learning module utilizes the results of all the computations performed during the previous iteration(s) aiming to simulate complex CMAQ processes (except the first iteration, which uses the inputs generated using random methods described in the section 4.2).

Currently, the learning module utilizes the following machine learning methods:

**Cell prediction method**
This method represents the basic learning approach – the learning module predicts every input cell independently. Thanks to this principle, every cell is predicted separately from all selected measurements. Suitable measurements are identified with geographic distance between measured and predicted cell or with high correlation value.

If suitable measurement is not found, the cell value is not predicted, but interpolated from its neighbours.

**Region prediction method**
Method of region prediction uses regions (see section 4.1) instead of independent area cells. The new value of the whole region is predicted at once and applies for every cell in the region uniformly.

Thanks to the region prediction principle, the computation is faster, because the number of predictions that have to be prepared has decreased. This method is also able to eliminate big prediction errors, because it is using all the suitable measurements of every region cell together, therefore the number of measurements used during such a prediction is often bigger and the prediction error becomes smaller.

**Inverse prediction method**
The method of inverse prediction is special, because unlike to the other methods that predict new inputs for the next iteration, this method works exactly opposite – it tries to predict outputs of CMAQ simulations for defined inputs. These predicted outputs can be compared against measurements and suitability of provided inputs can be measured. Compared to CMAQ simulations, predicted outputs are less precise, but prediction of thousands inputs takes less time than one CMAQ simulation.

This method often work with other mentioned methods – method of cell or region prediction predict new possible CMAQ inputs and method of inverse prediction tries to further improve this guess. To test more possible CMAQ inputs, the method of inverse prediction can also generate thousands of new inputs which can be fully random or based on actual predictions which have been made with other methods.
8 Application evaluation

In order to evaluate the ability of proposed application – and especially the ability of employed learning methods – to improve the initial BaP concentration in the soil making the simulations better correspond to real measurements, several tests were run using CMAQ input data (area and cell information, meteorological data, etc.) provided by the RECETOX laboratory (Research Centre for Toxic Compounds in the Environment).

While the CMAQ input data and initial soil BaP concentrations were sufficient for the evaluation, the real measurements – which the application tries to make the CMAQ simulations outputs better corresponding to (by predicting more precise soil concentrations inputs) – were not sufficient at all: usually, there is just a very limited number of real measurements available (in terms of density; the area of interest having 121 × 121 cells is usually covered by no more than 10 measured cells).

To overcome this issue and allow more complex evaluation of the employed learning methods (including, for example, their dependency on the number of measured cells), we decided to generate the “measured data” based on a single CMAQ simulation using the initial soil BaP concentrations (taken from the global model) as its soil input. This computation resulted in air BaP concentrations covering the whole area of interest, from which we later chose a random set of “measured cells” of the desired density.

However, this evaluation approach makes impossible to directly use the BaP soil concentrations provided by the global model as the input for the evaluation. Thus, the soil concentration input for the first iteration was also generated, employing the similar principles as described in the section 4.2. Using the described methods, we obtained a generated input having the relative error of 5.46E-006 (comparing the outputs of the CMAQ calculation using this input with the expected values of the “measured cells”).

Regarding the evaluation itself, all the methods of the learning module were tested with the following settings:

- the input data for CMAQ simulations covered the period between 1.7.2006 and 7.7.2006,
- the application run 15 iterations – during the first iteration, 100 CMAQ simulations (i.e., generated inputs for the first iteration) were run; the other iterations computed 5 CMAQ simulations,
- the number of “measured cells” used was 200 – even though bigger portion of measured cells could make the application behaving better, we wanted to make the evaluation close to reality (although, in reality, the number of measured cells is often even smaller),
- FANN library and its implementation of neural networks was used – neural network had 5 hidden layers and were trained during 5000 iterations or until 0.0001 error was achieved. Default algorithm (iPROP [5]) was used with FANN_SIGMOID (Equation 3) activation function for neurons in hidden layers and FANN_GAUSSIAN (Equation 4) for outputs neurons (all possible variants of activation function can be seen at [6])
Iterative Optimization of a Multi-compartmental Air Quality Modelling System

\[ y = \frac{1}{1 + \exp^{-2sx}}, d = 2s y (1 - y) \]  \hspace{1cm} (3)

\[ y = \exp^{-x^s x^s}, d = -2x y s, \]  \hspace{1cm} (4)

– during every iteration, the best relative error was computed as minimum from all the relative errors of every CMAQ simulation performed during the particular iteration. The relative error of a simulation was calculated (comparing its output with the values of the “measured cells”) using the simple formula depicted in the Equation 5.

\[ e = \frac{\sum_{i=1}^{n} |\text{measurement}_i - \text{CMAQ output}_i|}{\text{measurement}_i n} \]  \hspace{1cm} (5)

The Figures 5, 6 and 7 as well as the Table 1 present the evaluation results of the proposed application including details about all the learning methods used.

One can see that all the methods were able to significantly improve the initial input soil concentrations, which had the relative error 5.46E-006 – the best result achieved 7.61E-007 relative error, i.e. the initial input was improved by 7.17x. However, in all cases the highest improvement was achieved during the initial/first iteration, which run 100 CMAQ simulations and had the input data prepared and uniformly spread using random methods described in the section 4.2.

Unfortunately, it can also be seen that – even still providing some improvement – the employed learning methods do not sufficiently converge to the desired (i.e., measured) values: results of every method oscillate around initial guess during computation. This insufficient behaviour of the learning methods is probably caused by small portion of measured cells (the tested area of 121 \times 121 cells was covered by just 200 measured cells, i.e. 1.37%) and the limited number of iterations (resulting in small number of learning samples). Neural networks are also probably not really suitable machine learning algorithm with too slow convergence in such cases.

Furthermore, the region-based method could probably be improved by more sophisticated distribution of cells to regions: in the future, the region creation should use more information or these regions have to be prepared manually by experts.

<table>
<thead>
<tr>
<th>Method</th>
<th>Initial error</th>
<th>1. iteration error</th>
<th>Min. error</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell prediction</td>
<td>5.46E-006</td>
<td>8.30E-007</td>
<td>8.09E-007</td>
<td>6.75x</td>
</tr>
<tr>
<td>Cell + inverse prediction</td>
<td>5.46E-006</td>
<td>8.30E-007</td>
<td><strong>7.61E-007</strong></td>
<td><strong>7.17x</strong></td>
</tr>
<tr>
<td>Region prediction</td>
<td>5.46E-006</td>
<td>8.30E-007</td>
<td>8.24E-007</td>
<td>6.63x</td>
</tr>
</tbody>
</table>

Table 1: Comparison of initial guess and the minimal error achieved by the learning methods employed
9 Conclusions

The introduced application represents a complex system (described in more detail in [7]) that should be able to improve initial estimations of POP concentrations (currently focusing on BaP pollutant) in the soil in the future.

M. Tomáš et al.
Because of its computing demands (the CMAQ simulations are very complex and time-consuming), the application – currently implemented using Python and R language scripts – is designed to be able to utilize computing sources provided by supercomputing centres like MetaCentrum and CERIT-SC, allowing to run as many computations and iterations as possible. Its core is highly parallelized, therefore performed operations take as little time as possible.

The preliminary results show that the application and proposed learning methods are able to improve initial estimations of concentrations of POPs in the soil. Unfortunately, tests also demonstrate that although improving the initial guesses, the currently employed learning methods are not able to sufficiently converge during iterative computation. This could be caused by improperly chosen machine learning methods and/or insufficient number of used measurements. Especially, the choice of machine learning method represents the biggest challenge for future application improvement, and thus becomes the main focus of our future work. Also, further investigation of region-based method should be performed, since we still believe in their potential to provide far-better results.

Acknowledgement

Computational resources were provided by the MetaCentrum under the program LM2010005 and the CERIT-SC under the program Centre CERIT Scientific Cloud, part of the Operational Program Research and Development for Innovations, Reg. no. CZ.1.05/3.2.00/08.0144.

References

Part IV

Presentations
Randomized fuzzy formal contexts

Lubomír Antoni, Stanislav Krajčí, Ondrej Krídlo

Institute of Computer Science, Faculty of Science,
Pavol Jozef Šafárik University in Košice, Jesenná 5, 040 01 Košice, Slovakia.
lubomir.antoni@student.upjs.sk, stanislav.krajci@upjs.sk, ondrej.kridlo@upjs.sk

Formal concept analysis is a special data-mining method which generates the clusters of objects and attributes (called formal concepts) from the object-attribute tables (called formal contexts) [2]. The efficient selection of relevant clusters is an important issue and several studies have focused on this scalability question. A modified Rice-Siff algorithm represents a method for selecting the significant clusters with respect to metric properties [3] and the \( \alpha \)-cuts from fuzzy logic were applied to define the subset quality measure [5]. The stability index [4] is a plausibility measure for concept-based hypotheses. A high stability index signalizes that cluster does not disappear if the subset of attributes is modified. The phenomenon of the basic level of concepts is applied to select important clusters in [1].

In this work, we present a probabilistic approach to an issue of one-sided formal concepts stability. We define the randomized fuzzy formal contexts which are derived from the original one-sided formal context by adding the realizations of random variables with a normal distribution to each original value. We propose the Gaussian probabilistic index to compute the probability that some subset of objects is stable due to random fluctuation. We describe the algorithm and study the properties of the Gaussian probabilistic index. In combination with the modified Rice-Siff algorithm, a novel approach improves the analysis of the most relevant one-sided formal concepts.

The paper was accepted and presented at ICFCA 2015 (13th International Conference on Formal Concept Analysis).

References

1. R. Bělohlávek, M. Trnčka, Basic level of concepts in formal concept analysis, ICFCA 2012, pp. 28–44.

* This work was supported by the Scientific Grant Agency of the Ministry of Education, Science, Research and Sport of the Slovak Republic under contract VEGA 1/0073/15.
The Hanoi Omega-Automata Format

Tomáš Babiak1, František Blahoudek1, Alexandre Duret-Lutz2, Joachim Klein3, Jan Křetínský5, David Müller3, David Parker4, and Jan Strejček1

1 Faculty of Informatics, Masaryk University, Brno, Czech Republic
2 LRDE, EPITA, Le Kremlin-Bicêtre, France
3 Technische Universität Dresden, Germany
4 University of Birmingham, UK
5 IST Austria

Finite automata over infinite words, ω-automata, play a crucial role in formal verification. For instance, they are a key component in the automata-theoretic approach to LTL model checking, where the property in question is encoded as an ω-automaton. There is a long history of research and ongoing tool development, trying to produce more compact automata in theory and in practice.

Formats to represent ω-automata have mostly been defined in an ad-hoc manner, tailored to their particular tools, setting and scope, and tend to be restricted to a few specific acceptance conditions. The one format that covers most common acceptance conditions (Büchi, generalized Büchi, co-Büchi, Rabin, Streett, etc.) and automata structures (deterministic, non-deterministic, and alternating) is the XML-based Goal File Format (GFF) used internally by the Goal tool. It uses specific encodings for the different acceptance conditions. For instance, there is a special notation to define the sets in each acceptance pair of Rabin conditions. This necessitates changes to the format and its parsers when introducing new acceptance conditions and makes acceptance-agnostic manipulations difficult.

Based on our experience as implementers of tools producing, consuming, and manipulating ω-automata, we have set out to define a common, flexible, and extensible format for representing ω-automata in a uniform way. The result is the Hanoi Omega-Automata (HOA) format1 which was accepted to CAV 2015.

A crucial feature of the format is the introduction of a generic way to specify the acceptance condition as an arbitrary Boolean formula over the acceptance primitives “infinitely often” and “finitely often”, covering the common acceptance conditions used so far and more. More complex acceptance conditions can bring a significant gain in efficiency for some algorithms by using smaller automata, which was recently shown by Chatterjee et al. for probabilistic model checking.

The ultimate goal of the format is to be capable to express most types of ω-automata used and to-be-used in formal verification. This is achieved by support of the following:

- deterministic, non-deterministic, and alternating ω-automata,
- both state-labelled and transition-labelled ω-automata,

1 The discussion about this format started during ATVA’13 in Hanoi, hence the name.
The main features of the presented format are

- succinctness and human-readability,
- extensibility using header for additional information, and
- streaming support for processing automata in batches.

The full specification of the format and some examples of use can be found at http://adl.github.io/hoaf/ where it has been openly developed. We encourage other tool developers to report issues and suggest improvements there. Together with the specification we implemented support of the format in wide range of established tools and we already observed easier interactions of the tools. The representation of acceptance conditions allows a significant flexibility and performance increase, and encourages tool developers to expand the range of supported acceptance conditions.
Device-independent randomness extraction from an arbitrarily weak min-entropy source

Jan Bouda¹, Marcin Pawlowski³, Matej Pivoluska¹, and Martin Plesch¹,²

¹ Faculty of Informatics, Masaryk University
² Slovak Academy of Sciences, Bratislava
³ University of Gdansk

Randomness is an invaluable resource in today’s computer science. The need for high quality randomness (close to uniformly distributed and uncorrelated to any other existing data) is evident especially in the field of cryptography, where malfunctioning random number generators can cause catastrophic failures and lead to the total loss of security [4].

All the recent designs of random number generators (pseudorandom number generators, classical and quantum hardware generators), share one common feature – the test of their functionality is based on statistical tests, which in the light of recent events [5] might not be sufficient for cryptographic applications, as it does not capture the essential property required from cryptographically secure random number generators – the random numbers must be unpredictable to the adversary, not the user.

Surprisingly enough, this property is verifiable with the use of quantum mechanics. Bell type tests are well known tool in quantum information processing. These tests can verify, whether given non-communicating devices use quantum states and randomness generating measurements. In short, these tests are impossible to pass with pre-described lists of answers. The caveat is that challenges for the devices themselves need to be random, thus creating circular argument. However, the devices can produce more randomness than they consume, thus providing us with device independent randomness expansion.

In order to close the circular argument, several researchers attempted to solve the problem of perfect randomness production, suggesting ways to extract existing weak randomness with the use of untrusted quantum devices – i.e. to use imperfect randomness as an input to the Bell type tests and still argue that their outputs contain fresh randomness.

Since imperfect randomness can be characterized in various ways, there are two flavours of these so called device independent randomness extractors – with input characterized as a Santha-Vazirani source [3] or Min-Entropy source [2], [1]).

In our paper [1] we designed a protocol to extract randomness from min-entropy random sources. Another interesting angle this result can be viewed is the following – classically, the task of transforming a single weak source into a fully random bits is known to be impossible. However, with non-classical devices the task becomes possible.
Nested Antichains for WS1S

Tomáš Fiedor, Lukáš Holík
Ondřej Lengál, and Tomáš Vojnar

FIT, Brno University of Technology, IT4Innovations Centre of Excellence,
Czech Republic

Weak monadic second-order logic of one successor (WS1S) is a powerful, concise, and decidable logic for describing regular properties of finite words. Despite its nonelementary worst case complexity, it has been shown useful in numerous applications such as verification of pointer programs, deciding related logics or synthesis from regular specifications. Most of the successful applications were due to the tool MONA, which implements a finite automata-based decision procedure for WS1S and WS2S (a generalization of WS1S to finite binary trees).

The decision procedure of MONA works with deterministic automata; it uses determinization extensively and relies on minimization of deterministic automata to suppress this complexity blow-up. However, the worst case exponential complexity of determinization often significantly harms the performance of the tool. Despite many optimizations implemented in MONA and the other tools, the worst case complexity of the problem sometimes simply strikes back.

We propose a novel approach for coping with alternating quantification as the main source of nonelementary complexity of deciding WS1S formulae. Our approach is applicable within the state-of-the-art automata-based WS1S decision procedure implemented, e.g. in MONA.

The way in which the standard decision procedure processes quantifiers involves determinization, with its worst case exponential complexity, for every quantifier alternation in the prefix of a formula. Our algorithm avoids building the deterministic automata—instead, it constructs only those of their states needed for (dis)proving validity of the formula. It uses a symbolic representation of the states, which have a deeply nested structure stemming from the repeated implicit subset construction, and prunes the search space by a nested subsumption relation, a generalization of the one used by the so-called antichain algorithms for handling nondeterministic automata.

We have implemented our approach as the dWiNA tool and compared it with the state-of-the-art MONA tool. Our experimental results are encouraging (our tool often outperforms MONA) and show that the direction started in this paper—using modern techniques for nondeterministic automata in the context of deciding WS1S formulae—is promising.

This presentation is based on a paper with the same name that appeared in the proceedings of TACAS 2015.

Acknowledgement. This work was supported by the following projects: Czech Science Foundation (projects 14-11384S and 202/13/37876P), the BUT FIT project FIT-S-14-2486, and the EU/Czech IT4Innovations Centre of Excellence project CZ.1.05/1.1.00/02.0070.
Optimizing CUDA code by kernel fusion: application on BLAS

Jiří Filipovič, Matúš Madzin, Jan Fousek, and Luděk Matyska

1 Faculty of Informatics, Masaryk University, Brno, Czech Republic
{fila,gotti,izaak}@mail.muni.cz
2 Institute of Computer Science, Masaryk University, Brno, Czech Republic
ludek@ics.muni.cz

Contemporary accelerators, such as CUDA GPUs have significantly higher arithmetic throughput than a memory throughput. Thus many kernels with low flop-to-word ratio executed on GPUs are memory bound and cannot fully exploit the available arithmetic power. Examples of such memory-bound kernels are BLAS-1 (vector-vector) and BLAS-2 (matrix-vector) operations.

Kernel fusion can improve memory locality by placing shared data, originally passed via off-chip global memory, into a faster, but distributed on-chip memory. The fusion is difficult to perform for general kernels, however limiting the kernels to nested combinations of map and reduce enables us to perform the fusion automatically. However fusing all kernels together doesn’t always lead to performance improvement, therefore we show, how the search for the most efficient combination of partial fusions can be performed in a source-to-source compiler.

The compiler works with a library of application specific elementary function kernels, and a script describing semantics of particular computation. The compiler for given script creates, prunes and searches the optimization space of possible fusions and their combinations using performance prediction heuristic. The code of the selected implementation of the script is generated using the templates and metadata from the library of elementary function kernels.

We demonstrate the features of the compiler on sequences of BLAS-1 and BLAS-2 routines and evaluate the performance improvement by fusions. To represent these sequences in our compiler, we handle the (possibly large) vectors as lists of subvectors of fixed size, and matrices as nested lists of tiles. The BLAS routines are then expressed as nested applications of maps of arithmetic operators, and basic reduce functions such as sum. We compare the implementation generated by our compiler with CUBLAS—hand-tuned implementation of BLAS for CUDA GPUs. Many of the evaluated sequences are used frequently in performance-critical parts of numerical applications, and our compiler is able to achieve speedup up to 2.24x over the CUBLAS.

Algorithmic meta-theorems are general algorithmic results applying to a whole range of problems, rather than just to a single problem alone. Such results are some of the most sought-after in algorithmic research. Many prominent algorithmic meta-theorems are about model checking; such theorems state that for certain kinds of logic $L$, and all classes $C$ that have a certain structure, there is an algorithm that takes as an input a formula $\phi \in L$ and a structure $S \in C$ and efficiently determines whether $S \models \phi$. Here $S \models \phi$ is read as “$S$ models $\phi$” or “$\phi$ holds on $S$.”

In the past two decades the model checking problem for first-order (FO) logic received considerable attention. Since with the trivial brute-force algorithm it is possible to evaluate any FO formula $\phi$ in time $O(n^{|\phi|})$ (here $n$ is the size of the structure), the focus of research is on identifying classes of structures on which this runtime can be improved to $f(\phi) \cdot n^c$, where $f$ is any function and $c$ is a constant (algorithms with such runtime are called fixed-parameter tractable (FPT) algorithms).

The main focus of research in this direction have been sparse relational structures – culminating in the FPT algorithm by Grohe, Kreutzer and Siebertz for FO model checking of nowhere dense classes of graphs [STOC’14], with dense structures starting to attract attention only recently. Bova, Ganian and Szeider [LICS’14] initiated the study of the complexity of FO model checking on partially ordered sets (posets). They showed that model checking existential FO logic is FPT on posets of bounded width, where the width of a poset is the size of the largest antichain in the poset. The existence of an FPT algorithm for general FO model checking on posets of bounded width, however, remained open. We resolve this question in the positive by giving an algorithm that takes as its input an $n$-element poset $P$ of width $w$ and an FO logic formula $\phi$, and determines whether $\phi$ holds on $P$ in time $f(\phi, w) \cdot n^2$.

The presented results were accepted to FOCS 2015 and were obtained together with Petr Hliněný, Daniel Lokshtanov, Jan Obdržálek, Sebastian Ordyniak, M.S. Ramanujan and Saket Saurabh.
Search-Space Partitioning for Parallelizing SMT
Solvers

Matteo Marescotti, Antti E. J. Hyvärinen, and Natasha Sharigina
Faculty of Informatics, University of Lugano
Via Giuseppe Buffi 13, CH-6904 Lugano, Switzerland

The Satisfiability Modulo Theories (SMT) problem is the decision problem of determining whether a propositional formula is satisfiable, given that some of the variables have an interpretation with respect to background theories. The expressiveness of SMT makes it suitable for a vast range of application domains, and for that reason it has recently attained significant interest from both industry and academia.

The computational cost of SMT can be very high given that just the problem of determining the propositional satisfiability, the Boolean Satisfiability Problem (SAT), is proven to be NP-complete, and the introduction of the background theories can only make the problem harder. Nevertheless relatively little research exists on how parallel computing can be used to speed up SMT solving.

Two important approaches can be used to parallelize SMT solving: algorithm portfolio and the divide-and-conquer approach.

Algorithm portfolio is widely used because the heuristics that guides the solver’s choices are intrinsically inaccurate, and small changes can result in significant differences in run times. In this work we introduce such changes by randomly initializing the seeds that each solver of the portfolio uses. However algorithm portfolio alone seems to hit a scalability limit where adding more CPUs does not provide more speed-up.

The divide-and-conquer approach partitions the search-space of the original instance: this results in many instances all different from each other. This approach leads to an inherent problem when the original instance is unsatisfiable: we must wait for all the instances to be solved in order to prove the unsatisfiability of the original one. Those instances, despite being usually easier than the original one, still might be challenging, and since we have to wait for the "unluckiest" one to be solved, an exclusive use of this approach can lead to worse performances.

In this work we address the challenge of parallelizing SMT solving introducing an abstract algorithmic framework called parallelization tree. The parallelization tree can be used to combine the two parallel approaches in order to exploit the positive aspects of both. We show experimentally, that several instantiations of the parallelization tree framework lead to an increment on the number of instances solved within a given amount of time.

All the experimentations have been made using the solver OpenSMT2, developed at the University of Lugano, that is capable of handling the theory of quantifier free uninterpreted functions with equalities (QF$_{UF}$). We run all the experiments in a cluster consisting of nodes with two AMD quad-core Opteron
2344 HE CPUs, and each node was running at most four solver instances. We used 16 nodes, hence 64 solver instances running concurrently.

We noted that the best instantiation of the parallelization tree is the one that divides the main instance in 8 instances, each of those solved with a portfolio made of 8 solvers; this approach also provides a general and significant speed-up on the other instances. Among our benchmark set, this parallel approach was able to solve 6 more instances within the timeout. The article with all the details has been accepted at the conference SAT2015 [1].

Acknowledgements. This work was financially supported by SNF project number 153402.

References

Large-scale Ultrasound Simulations Using the Hybrid OpenMP/MPI Decomposition

Jiri Jaros¹, Vojtech Nikl¹, and Bradley E. Treeby²

¹ Faculty of Information Technology, Brno University of Technology, Bozetechova 2, 612 66 Brno, CZ
{jarosjir, inikl}@fit.vutbr.cz

² Dept. of Medical Physics and Biomedical Engineering, University College London, Malet Place Eng Bldg, London WC1E 6BT, UK
b.treeby@ucl.ac.uk

1 Abstract

The simulation of ultrasound wave propagation through biological tissue has a wide range of practical applications. Recently, high intensity focused ultrasound has been applied to functional neurosurgery as an alternative, non-invasive treatment of various brain disorders such as brain tumours, essential tremor, and Parkinson’s disease. Performing accurate ultrasound simulations, however, requires the simulation code to be able to operate on large domains and deliver the results in a clinically meaningful time.

The k-Wave toolbox is designed to simulate ultrasound wave propagation in soft-tissues and bone, modelled as fluid and elastic media, respectively. In the k-Wave toolbox, the k-space pseudospectral method is used to solve the system of governing equations. These equations are derived from the mass conservation law, momentum conservation law, and an empirically derived acoustic pressure-density relation that accounts for acoustic nonlinearity, absorption, and heterogeneity in the material properties.

This paper presents an improved implementation that exploits a 2D hybrid OpenMP/MPI decomposition. The 3D grid is first decomposed by MPI processes into slabs. The slabs are further partitioned into pencils assigned to threads on demand. The performance was investigated by a few simulation cases calculating the propagation of nonlinear waves in heterogeneous and absorbing media with a source driven by a sine wave.

For a domain size of $1024^3$, the hybrid code using 8192 cores enables the simulations to be accelerated by a factor of 4 compared to the pure-MPI code. Deployment of the hybrid code has the potential to eventually bring the simulation times within clinically meaningful timespans, and allow detailed patient specific treatment plans to be created.

Characterizing Multipartite Entanglement without Shared Reference Frames

Claude Klöckl\textsuperscript{12} and Marcus Huber\textsuperscript{13}

\textsuperscript{1} Universitat Autonoma de Barcelona, 08193 Bellaterra, Barcelona, Spain
\textsuperscript{2} Institute for Quantum Optics and Quantum Information (IQOQI), Austrian Academy of Sciences, Boltzmanngasse 3, 1090 Vienna, Austria
\textsuperscript{3} ICFO-Institut de Ciencies Fotoniques, 08860 Castelldefels, Barcelona, Spain

Classical Complexity theory describes tasks by the consumption of time and space resources. Quantum Information Theory introduces a new non classical resource: Entanglement. This resource is generally considered to be a vital ingredient for quantum protocols, that allow a speed up over classical protocols. However to certify the presence of entanglement in a system is a non trivial task and only possible by the development of certain inequalities called entanglement witnesses. If the results of measurements taken on a quantum mechanical system violate an entanglement witness, the presence of entanglement in the system can be inferred.

We want to present to you our work published in Phys. Rev. A 91, 042339 (2015). In this work we introduce an entanglement witness based on correlation tensor norms. These witnesses posses desirable theoretical properties that simplify the measurement process by abandoning the common requirement of a shared reference frame between measurements. Furthermore the authors believe that these witnesses could be constructed in an efficient way by the employment of classical machine learning techniques. We would be happy to discuss with the audience in how far a collaboration between the distinct fields of quantum information and machine learning could be fruitful.
Optimizing Job Scheduling in National Grid Computing System: Theory and Practice

Dalibor Klusáček¹,² and Šimon Tóth¹ and Gabriela Podolníková¹

¹ Faculty of Informatics, Masaryk University, Brno, Czech Republic
² CESNET a.s., Prague, Czech Republic
{xklusac,toth,xpodol}@fi.muni.cz

In the past two decades many new scheduling approaches have been proposed in order to optimize job scheduling in large computational systems. However, instead of being tested on real systems, these works often remained on the “theoretical level”, using simplified problem models together with a simulator. As a result, common mainstream job schedulers are still using only a limited set of scheduling policies. For example, the so called backfilling policy (an optimization of First Come First Served designed to increase resource utilization) was introduced in 1995 and still remains one of the most advanced scheduling solutions today. It illustrates the obvious gap between theory and practice — real systems are hard and require robust, complex solutions.

In this work [1] — recently presented at the Job Scheduling Strategies for Parallel Processing — we explain that the problem of optimizing the performance of an actual production scheduler is far more complex than just choosing a proper scheduling algorithm. Using our experience from the Czech National Grid Infrastructure MetaCentrum, we show that production systems need to balance around a set of policies that are established to satisfy both resource providers and users, e.g., a fair-sharing policy guaranteeing fair resource allocation, as well as proper queue setup, guaranteeing reasonable distribution of system resources among various job classes. Most importantly, our work [1] provides both the theoretical analysis covering impact of various system policies as well as the practical evaluation using data gathered from the real production system.

The strength of this study is based on the fact that the presented solution has been applied in practice within the Czech National Grid computing system MetaCentrum. We demonstrate that our solution — including the improved queue configuration and the new fair-sharing algorithm — largely improves both the fairness as well as the overall system performance with respect to the previously used scheduling approach.

References


⋆ This work was kindly supported by the LM2010005 project funded by the Ministry of Education, Youth, and Sports of the Czech Republic and by the grant No. P202/12/0306 funded by the Grant Agency of the Czech Republic.
Toward natural multi-user interaction in advanced collaborative display environments

Vít Rusňák¹, Lukáš Ručka¹, and Petr Holub¹,²

¹ Masaryk University Botanická 68a, Brno, Czech Republic
² CESNET z.s.p.o. Zikova 4, Praha, Czech Republic,
xrusnak@fi.muni.cz, xrucka.fi.muni.cz, hopet@cesnet.cz

Scalable resolution displays (SRDs) have become widely spread in research centers, laboratories, and public spaces during the last decade. There are continuous efforts in transformation SRDs from passive screens to interactive environments where users can interact independently via multi-touch surfaces, in-air interaction or pointing devices. Continuous user tracking and association of input events with users, which could considerably improve user experience, is still a largely unexplored topic.

In this paper, we present the concept of cue-less multi-user interaction in environments where horizontal or vertical SRDs are in the role of central visualization platforms. Next, several techniques enabling unobtrusive user tracking and their association with input events are described. Global coordinate space describes the whole area where users can interact with the visualization system. It is constructed from viewports. These represent sensor devices present in the environment—e.g., RGB camera, MS Kinect, multi-touch panel. Virtual multi-touch overlay provides a scalable approach for construction of multi-touch input layer from multiple sensors of the same kind. It behaves as a single seamless sensor. Computation overhead such as concatenation of gesture strokes performed over several physical sensors is hidden. Next, we present top-view and rear-view unobtrusive user-tracking algorithms suitable for commodity depth sensors.

Further, we introduce an open-source framework implementing these techniques. The framework can serve as a prototyping and evaluation platform for new interaction techniques and their comparison. Besides, it could be utilized for deployment of interaction layer in interactive environments based on SRDs. We provide the evaluation of the framework prototype implementation. The focus is laid on the accuracy of user tracking and association with input events as well as on processing speed of proposed algorithms. The results confirm feasibility of proposed approach. The prototype implementation provides high accuracy (over 90%) in tracking algorithms while ensuring real-time sensor data processing.

The paper was published in Future Generation Computer Systems (FGCS) journal in 2015. The paper is available at: http://dx.doi.org/10.1016/j.future.2015.03.019.
Optimized SLA Assured Service Brokering (SLaB) and Service Verification in Multi-Cloud Environment

Shyam S. Wagle

University of Luxembourg, 6, rue R. Coudenhove-Kalergi, Luxembourg City, Luxembourg,
shyam.wagle.001@student.uni.lu

Concept of cloud brokering facilitates Cloud Service Users (CSUs) to discover cloud services according to their actual service requirements. Cloud service providers (CSPs) commit cloud services to the CSUs through Service Level Agreement (SLA). SLA is composed of different Quality of Service (QoS) rules, which are obligations that have to be followed by the CSPs.

In the current practice, cloud service broker (CSB) selects the cloud services from multi-cloud environment according to SLA offered by Cloud CSPs. Difficulty in selecting cloud services from multiple CSPs is; there is no guarantee that CSPs deliver cloud services according to SLA offer. In our observation, it is found that most of the CSPs do not fulfill the service commitment mentioned in SLA agreement. It is necessary to be ensured that CSPs are delivering cloud services according to SLA commitment before recommending the cloud services to the CSUs. There is no or very little practice that CSU or CSB consider the service delivery status of the cloud providers while selecting the appropriate cloud services because there is a lack of verification framework to provide the service delivery status of the CSP.

In this presentation, we propose a service verification model of service delivery status of commercially available CSPs. As CSUs’ requirements are independent to each other and different. For instance, normally, CSU prefer minimum Cost, maximum Availability, maximum Performance, maximum Security etc. for cloud services. It is not possible always to provide the services with high performance in minimum Cost. Some CSUs prefer cloud services with high performance even Cost is high. So, in such cases there is not a single objective and not a single optimum solution. To provide the best cloud services according to CSUs’ requirements, we propose Pareto solutions that consider multiple criteria (delivered SLA performance) to provide the optimum set of solutions to the CSUs from multiple CSPs.

References


1 Initial conceptual work of this paper has been published in 7th IEEE/ACM International Conference on Utility and Cloud Computing [1]
Author Index

Antoni, Ľ, 85
Babiak, T., 86
Blahoudek, F., 86
Bouda, J., 88
Duret-Lutz, A., 86
Efstathiou, Ch., 70
Fiedor, T., 89
Filipovič, J., 90
Fousek, J., 15, 90
Gajarský, J., 91
Holík, L., 89
Holub, P., 97
Huber, M., 95
Hyvärinen, A. E. J., 92
Ištvánek, J, 58
Jaroš, J., 94
Klöckl, C., 95
Klein, J., 86
Klusáček, D., 96
Kocina, F., 28
Krajčí, S., 85
Křenek, A., 58
Král, O., 85
Kučera, J., 36
Kunovský, J., 28
Lengál, O., 89
Müller, D., 86
Madzin, M., 90
Marescotti, M., 92
Matyska, L., 90
Meduna, A., 36
Mic, V., 45
Nečasová, G., 28
Nikl, V., 94
Novak, D., 95
Parker, D., 86
Pawlowski, P., 88
Pivoluska, M., 88
Plesch, M., 88
Podolníková, G., 96
Rebok, T., 70
Řepková, J, 58
Ručka, L., 97
Rušnák, V., 97
Sharygina, N., 92
Šimek, M., 58
Soukup, O., 36
Strejcek, J., 86
Svobodová Vařková, J., 58
Tomáš, M., 70
Tóth, Š., 96
Treeby, B. E., 94
Veigend, P., 28
Vojnar, T., 89
Wagle, S. S., 98
Zezula, P., 45
Organisers

Faculty of Informatics
Masaryk University
Botaničká 68a
Brno, Czech Republic
http://www.fi.muni.cz

Faculty of Information Technology
Brno University of Technology
Božetěchova 2
Brno, Czech Republic
http://www.fit.vutbr.cz

Workshop Sponsor

Zoner
MEMICS 2015
Tenth Doctoral Workshop on Mathematical
and Engineering Methods in Computer Science

Printing and publishing: LITERA, Tábor 43a, 612 00 Brno
Edition: first
Year of publication: 2015

Typesetting: camera ready by paper authors and PC members, data conversion
and design by Jaroslav Rozman

Cover design: Tomáš Staudek