Big Data Spectra Analysis Using Analytical Programming and Random Decision Forests

Petr Šaloun[1], Peter Drábik[1], Ivan Zelinka[1], and Jaroslav Bucko[2]

[1] VŠB-Technical University of Ostrava FEE&I, 17. listopadu 15/2172
CZ-708 33 Ostrava-Poruba, Czech Republic
[2] Slovak Technical University Bratislava FIIT, Ilkovičova 2,
SK-84216 Bratislava 4 Slovakia
{petr.saloun,peter.drabik.st,ivan.zelinka}@vsb.cz,
bucko.jaroslav@gmail.com

Abstract. Spectra analysis on large datasets is in focus of this paper. First of all we discuss a method useful for spectra analysis – analytical programming and its implementation. Our goal is to create mathematical formulas of emission lines from spectra, which are characteristic for Be stars. One issue in performing this task is symbolic regression, which represents the process in our application, when measured data fits the best represented mathematical formula. In past this was only a human domain; nowadays, there are computer methods, which allow us to do it more or less effectively. A novel method in symbolic regression, compared to genetic programming and grammar evolution, is analytic programming. The aim of this work is to verify the efficiency of the parallel approach of this algorithm, using CUDA architecture. Next we will discuss parallel implementation of random decision forest (RDF) to classify huge amounts of various spectra. The mathematical formulas obtained via AP will be used to reduce attributes of explored spectra. Our goal is to propose scalable algorithm for classification of such data, which will preferably need only one pass over data, while maintaining acceptable accuracy. Later we will try to create module compatible with VO and DAta Mining and Exploration project.

Keywords: analytical programming, spectra analysis, CUDA, evolutionary algorithm, differential evolution, parallel implementation, symbolic regression, random decision forest, data mining, virtual observatory.

1 Introduction

Nowadays, astronomy is one of the scientific disciplines which produce enormous amounts of data per day, which needs to be processed automatically. With advancement of surveillance methods the data grows in amount and complexity. In our work we focused on the spectra analysis of Be star-candidates. Be stars are hot B-type stars (effective temperature 10 000 K to 30 000 K) with luminosity class III to V (i.e. not supergiant stars) whose spectrum has shown at least

K. Saeed and V. Snášel (Eds.): CISIM 2014, LNCS 8838, pp. 266–277, 2014.

once an emission line – usually hydrogen in the Balmer line, see Figure 1. Sometimes, other emission lines are visible, for example neutral helium. Even when the spectrum goes back to *normal*, the star remains in the Be star class [9]. Some of them are among the brightest stars in the sky [2]. With the high complexity, the data attributes need to be reduced to less dimensions. The spectra attributes will be reduced from value for every observed wavelength in spectra to lines in certain wavelength ranges, which represent features of object the spectra belongs to.

Non-image data, such as spectra, are also mainly distributed in astronomy in Flexible Image Transport System (fits) format, standardized in 1981 [1]. This format can have the `*.fts`, `*.fits`, `*.fit` extensions and a major feature it has is that images metadata are store in a human readable ASCII header. In our work we processed fits format data files with fv FITS Editor, available from the standard Ubuntu repository.

Astronomical data are obtainable from various virtual observatories (VO). A set of standards to access and manipulate such data was created by International Virtual Observatory Alliance. They describe API to communicate with VO web services. The services provide not only FITS files, but VOTable formatted data too.

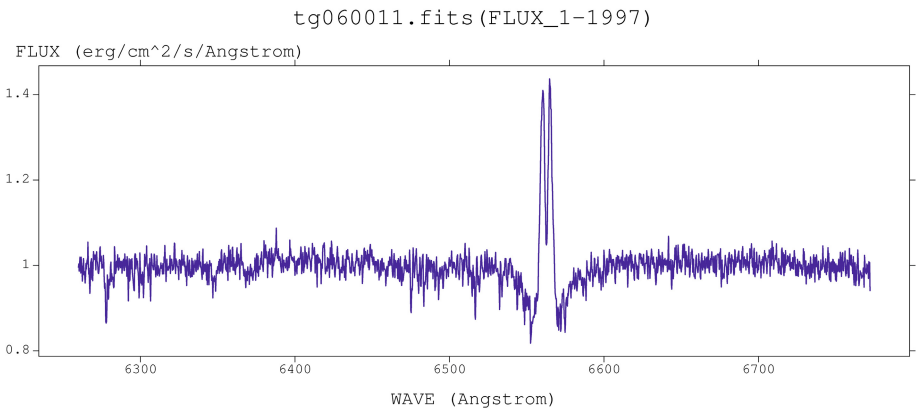

Fig. 1. An example of spectra with a characteristic Be star emission

2 Data Mining with Random Decision Forests, an Introduction

Random decision forests are a data mining structures proposed in 2001 by Leo Breiman [14] for supervised classification. Random decision forests are ensembles of classification and regression trees. They main difference from other decision trees and ensembles is a way of splitting nodes in trees to make decisions.

2.1 Training

During training the bagging is used. Each tree in ensemble is grown using subset consisting of cases, drawn random from training set. From this part of training data, a random subset of features is selected. Node split is selected randomly from best splits. The trees are not pruned, unlike in other classification trees. The trees grown during training can be tested for error using out-of-bag estimates.

2.2 Prediction

During prediction the sample is classified by every tree in the forest. After that, the final class of sample is class with the most "votes" from trees as shown in Figure 2, or can by selected from mean of probabilities of classes from each tree.

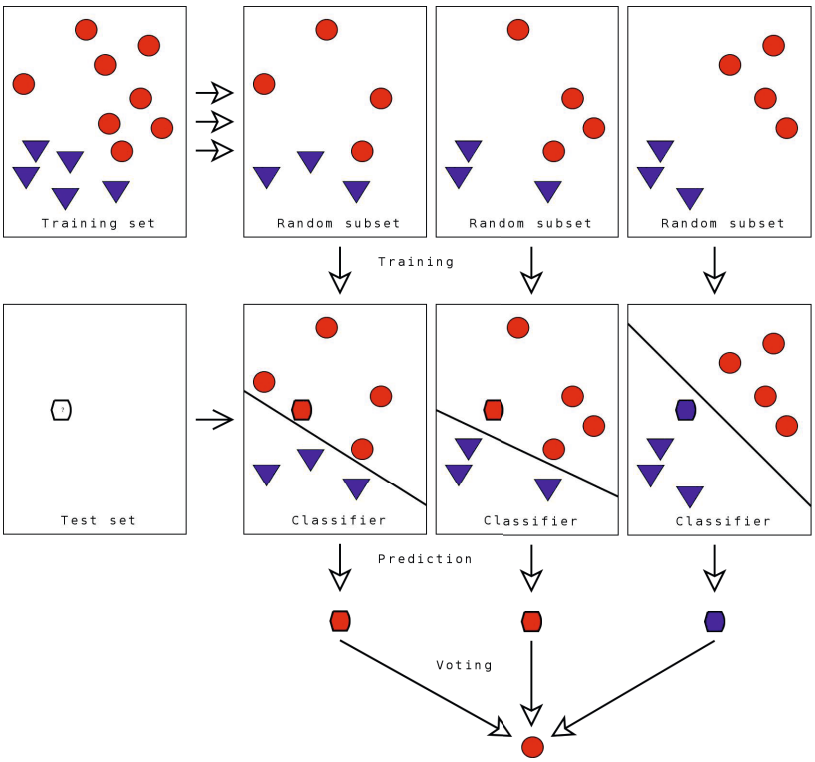

Fig. 2. Classification with bagging

2.3 Advantages of Random Forests

Random selection of features brings following advantages to RDF:

– robustness to noise and outliers,

– faster than bagging and boosting,
– simply and easily parallelized,
– no over-fitting.

RDF is considered one of the best learning algorithms, by its performance [16,17].

2.4 Scalable Implementation

Parallel implementation of RDF seems to be simple, by growing multiple trees at a time during training stage, and running prediction for multiple trees in forest at prediction stage. An optimal ratio between trees in ensemble and number of computing units should be found to minimize idle time waiting for new spectra samples to be loaded into memory. Since the working data is very large, a single pass over data is very time consuming. Everything should be done in that single pass, or at least in as few passes as possible. This makes decisions for best node split difficult. For this a histogram of values can be used like proposed by Ben-Haim for streaming parallel decision trees in 2010 [18] and used by Li [19]. In spectra analysis it is not necessary to worry about missing values, because missing value in astronomical spectra indicates, presence of certain chemical elements in object, what can be treated as another attribute. When not classifying spectra but other data, the missing values can be handled by substitution to median of values from histogram too. There are currently some scalable solutions for random forests:

Apache Mahout: is a data mining library based on Hadoop written in Java. With random forests allows the use of multiple classifiers.

Scalable random forest by Li B.: is a random forest implementation with MapReduce. It grows trees breadth-first. Uses histograms to calculate best split [19].

H2O: is a scalable machine learning platform, which uses L. Breimans implementation of random forests. It is capable to run on Apache Hadoop and Amazon EC2 (beta). Written in pure java.

2.5 Data Complexity Reduction

To improve speed and accuracy, data complexity reduction and relevant features of spectra selection is needed. The data reduction can be made with the help analytical programming by identifying interesting features and how they are represented in spectra and describing them by mathematical functions. It will not be necessary to break every observed spectra to set of features, instead, in every tree node, the spectra can be compared against one of the created mathematical functions and based on result, the decision of which child node descend to can be made.

3 Virtual Observatory

The Virtual Observatory is an international astronomical community-based initiative. It aims to allow global electronic access to the available astronomical

data archives of space and ground-based observatories and other sky survey databases. To obtain data from VO we must follow IVOA standards for data access layer. DAL services are provided as HTTP REST web services. For spectra analysis are interesting following services:

Simple Spectral Access Protocol defines a uniform interface to remotely discover and access one-dimensional spectra and aggregations of 1D spectra. Discovered datasets are returned in VOTable format,

Simple Line Access Protocol defines interface to spectral lines search from Spectral Line Data Collections. Spectral lines are returned in VOTable format.

Simple Image Access Protocol allows retrieving on-the-fly created images of sky given the position and size of the desired output image. After client has described image, service returns list of possible images in VOTable format.Client then chooses image to retrieve. Retrieved image is stored in FITS format.

Simple Cone Access Protocol allows to retrieve records from a catalogue of astronomical sources. The query describes sky position and an angular distance, defining a cone on the sky. The response returns a list of astronomical sources from the catalog whose positions lie within the cone, formatted as a VOTable.

We will use these protocols to obtain spectra and metadata of spectra to process.

4 DAta Mining and Exploration

DAME[1] is an Italian project to provide Astrophysics community with easy to use data mining suit. This suit is called DAMEWARE. Its focus is on processing of massive data sets with machine learning methods. It is based on S.Co.PE.[2], a general-purpose supercomputing infrastructure of the University of Naples Federico II. DAMEWARE model library is extendable via plugins. The only requirement is to have executable of model and to supply parameters for model or configuration files without need to know underlying DAMEWARE architecture. We would like to create a model plugin from our implementation.

5 Overview of Analytical Programming

Analytical programming, proposed in 2005 [4], was inspired by Hilbert spaces and genetic programming. The principles and general philosophy behind analytical programming (AP) stem from these two methods. Into AP an idea about the evolutionary creation of symbolic solutions is taken from GP while from Hilbert spaces the idea of functional spaces and the building of the resulting function by means of the search process are adopted in AP. The core of AP is based on a

[1] `http://dame.dsf.unina.it`
[2] `http://www.scope.unina.it`

set of functions, operators and so-called terminals, which are usually constants or independent variables as well as in GP and GE. The main aim of AP is to synthesize a suitable program which would fit the measured data as well as possible (with the given precision). For this reason, a discrete set handling (DSH) idea [7,8] was adopted in AP [5]. Discrete set handling creates an interface between the Evolutionary Algorithm (EA) and the problem. Therefore, we can use in AP almost any evolutionary algorithm. The individual is represented as a nonnumeric value and a numerical value is added to the evolutionary process as an integer index. This index represents an individual from the General Function Set (GFS).

5.1 Versions of AP

There are three versions of AP. AP_{basic} is a basic version of AP and uses constants from the terminal set. AP_{meta} – in this version there are constants not defined in the terminal set; in the terminal set there is only one general constant K and every constant K is estimated by a different or the same evolutionary algorithm. We can mention one disadvantage of this version – because of running evolution under evolution; it could be very slow for a large number of steps. There are a big number of evaluations of the fitness function. The last version is AP_{nf} – constants are estimated by non-linear fitting algorithm.

5.2 General Function Set

The GFS is a set of all mathematical objects – functions, operators and terminals. GFS consists of functions with different numbers of arguments. GFS is user-defined, so the content may differ. We must split the content of GFS into classes based on the numbers of arguments: 0_{args} are terminals, 1_{args} $(sin, cos, tan, \ldots)$, 2_{args} $(+, -, *, /, \ldots)$, etc. Choosing the right set may have a large influence on the convergence of AP.

5.3 Evolutionary Algorithm

AP was designed to be a very robust method and we can use almost any evolutionary algorithm. Individual steps, such as mutation, crossover, etc. are fully handled by the chosen evolutionary algorithm. Operations which make EAs when the algorithm is run do not have any influence on performance. Thus the result performance depends mainly on the correct choice of GFS individuals. For every evolutionary algorithm, the main goal is generally to reduce the fitness value to below a user-defined threshold, or to achieve maximum number of migrations for a Self Organizing Migrating Algorithm (SOMA), or in the case of Differential Evolution (DE) – generations.

5.4 Mapping Operators

Mapping is the phase when an individual is transformed into a useful mathematical function. It consist of two parts, DSH and security functions, to exclude the

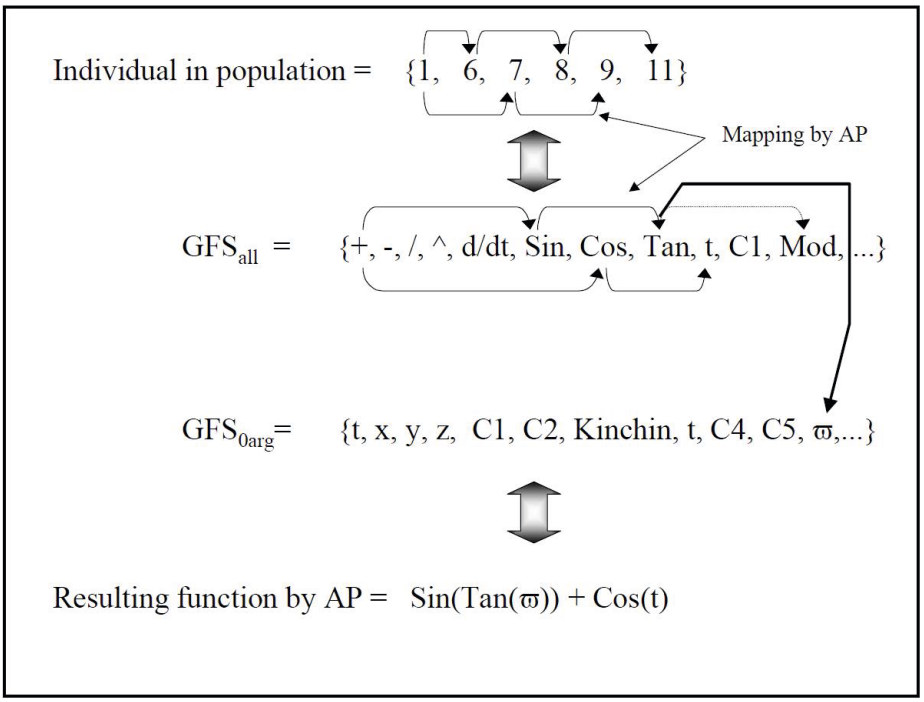

Fig. 3. Schema of mapping and security principles. Because of measuring distance to end of expression is *tan* replaced by ω [4]

creation of pathological individuals. In the evolutionary algorithm the individual is represented by a vector of indexes and is remapped to mathematical objects from GFS.

5.5 Reinforced Evolution

By running evolution, more or less suitable individuals are created, so one very good idea is to include the best individual as the terminal in GFS. The main idea of reinforcement is based on the addition of a just synthesized and partly successful program in an initial set of terminals [3]. The decision on whether the best value will be added to GFS is ensured by a user defined threshold value. If it is reached, from that moment it is added in GFS as a terminal, best solution and updated whenever a better solution with lower fitness is found.

5.6 Security Procedures

Evolution can result in an expression, which is not mathematically correct, thus we create a pathological individual (without argument). We can prevent this by distributing GFS into classes ordered by the number of arguments. By mapping

from evolution space to GFS, we measure the distance to the end of the expression. When the number of arguments is greater than the distance to the end of the expression, we choose individuals from the lower class. We must also pay attention so as to exclude errors from the fitness function, such as division by zero, functions with an imaginary or real part (if not expected), frozen functions (an extremely long time to get a cost value), etc. [3].

6 Parallel Implementation Using CUDA – An Overview

CUDA (Compute Unified Device Architecture) is a platform for parallel computing, developing by NVIDIA. CUDA is SIMT (Single Instruction Multiple Threads). Its main feature is that one instruction is executed by thousands of threads. However, kernels can effectively perform only basic operations. In the CUDA device various types of memory reside, as you can see in Table 1. Note: registers are the fastest memory on the GPU.

Table 1. CUDA Memory Types and Characteristics [13], (u.c. ≡ unless cached)

Memory	Location	Cached	Access	Scope
Register	On-chip	No	Read/Write	One thread
Local	On-chip	Yes	Read/Write	One thread
Shared	On-chip	N/A	Read/Write	All threads in a block
Global	Off-chip (u.c)	Yes	Read/Write	All threads + host
Constant	Off-chip (u.c.)	Yes	Read	All threads + host
Texture	Off-chip (u.c.)	Yes	Read/Write	All threads + host

The idea of implementing AP on parallel architecture is based on successful parallel implementation of evolutionary algorithms on CUDA, where a significant speeding up was achieved [10,11]. We were inspired to implement a parallel version of AP on CUDA, because of the good results achieved by the evolutionary algorithms when implemented on CUDA, although execution of this number of operations and structures may lead to worse performance results, as expected.

7 Testing Parallel Implementation – Methods, Datasets, Results

We cut the spectra from Figure 1 and obtained only the emission line in Figure 5 with 50 equidistant points, to satisfy our requirements. The data on the Be stars spectra come from the archive of the Astronomical Institute of the Academy of Sciences of the Czech Republic[3].

[3] Available from: `http://astropara.projekty.ms.mff.cuni.cz/spectra/newest/`

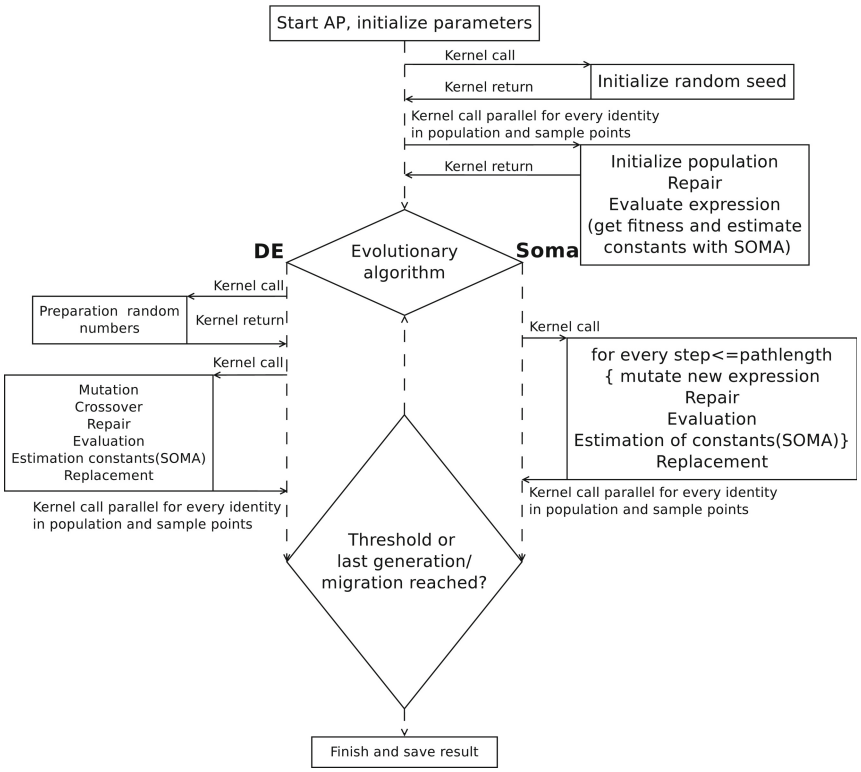

Fig. 4. The flowchart of AP implementation

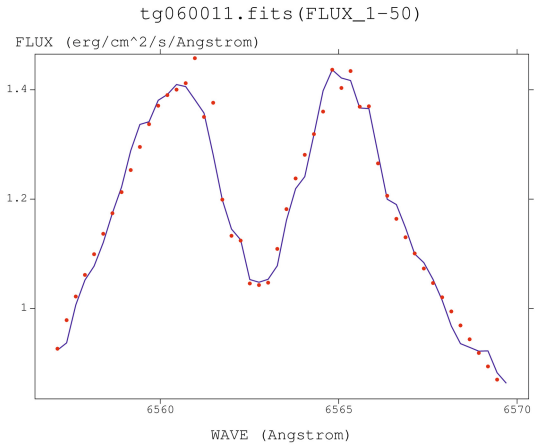

Fig. 5. Extracted emission line (blue) of Be star and our best result (red dots) of the SOMA algorithm with deterministic chaos

To be able to carry out a comparison in our tests we used a classical pseudo-random number generator and on the other site we tried to compare it to implemented deterministic chaos, with setting $A = 4$, which equation is as follows:

$$x_{n-1} = Ax_n(1 - x_n)$$

We carried out tests with SOMA and DE as the main evolutionary algorithm of AP. For estimating constants we chose the SOMA algorithm. The measured results are shown in Tables 2 and 3.

The settings for our implementation are shown in Table 2.

Table 2. Algorithm settings

DE		SOMA		SOMA constants	
NP	500	PopSize	500	PopSize	10
Dimensions	40	Dimensions	20	Dimensions	20
Generations	100	Migrations	10	Migrations	5
F	0.9	PRT	0.1	PRT	0.1
CR	0.5	PathLength	4	PathLength	3
		Step	0.21	Step	0.25

Table 3. Minimum, average and maximum fitness achieved from our tests. Each method was tested 10 times.

	DE PRNG	DE Chaos	SOMA PRNG	SOMA Chaos
MIN	4.02327	4.00432	0.85327	0.83326
AVG	4.30381	4.27877	1.41000	1.39508
MAX	4.57460	4.51520	1.84721	1.91367

Table 4. Execution of AP: with SOMA 95238096 evaluations of the cost function in the main EA, with DE 100000000

[s]	DE PRNG	DE Chaos	SOMA PRNG	SOMA Chaos
MIN	2688.59	1962.53	3161.47	2761.57
AVG	2805.84	2296.42	3305.62	2942.40
MAX	2873.77	3037.32	3372.46	3098.13

Thanks to the use of deterministic chaos against a pseudo-random number generator (PRNG) we speed up the running time by using simple mathematical operations, which is good for the CUDA kernel. Convergence of the fitness value is also a bit better.

8 Conclusion

We successfully implemented AP on CUDA and obtained relevant results. Both PRNG and deterministic chaos are suitable for running with parallel implementation of AP. The use of deterministic chaos seems to be a bit better. This is given by the fewer and simpler math operations needed to get a value. CUDA is not designed to run as long kernels, as we implemented, but there is a lot of room for future optimization of these kernels. We use lots of registers in the kernel, so performance falls from the ideal. In feature CUDA architecture, kernel under kernel will be probably run, which should bring a large improvement in performance. We want to compare our results with another parallel approach in the near future. On graphics hardware it will be Opencl, and OpenPM as a CPU variant respectively.

Acknowledgement. The following grants are acknowledged for the financial support provided for this research: Grant Agency of the Czech Republic – GACR P103/13/08195S, is partially supported by Grant of SGS No. SP2014/159, VŠB–Technical University of Ostrava, Czech Republic, by the Development of human resources in research and development of latest soft computing methods and their application in practice project, reg. no. CZ.1.07/2.3.00/20.0072 funded by Operational Programme Education for Competitiveness, co-financed by ESF and state budget of the Czech Republic.

References

1. Wells, D.C., Greisen, E.W., Harten, R.H.: FITS: A Flexible Image Transport System. Astronomy and Astrophysics Supplement Series 44, 363–370 (1981)
2. Rivinius, T., Carciofi, A.C., Martayan, C.: Classical Be stars. The Astronomy and Astrophysics Review, 1–86 (2013), doi:10.1007/s00159-013-0069-0
3. Zelinka, I., Davendra, D., Senkerik, R., Jasek, R., Oplatkova, Z.: Analytical Programming – a Novel Approach for Evolutionary Synthesis of Symbolic Structures. In: Kita, E. (ed.) Evolutionary Algorithms. InTech (2011), doi:10.5772/16166, ISBN: 978-953-307-171-8
4. Zelinka, I., Oplatkova, Z., Nolle, L.: Analytic Programming – Symbolic Regression by Means of Arbitrary Evolutionary Algorithms. Special Issue on Inteligent Systems, International Journal of Simulation, Systems, Science and Technology 6(9), 44–56 (2005) ISSN 1473-8031
5. Zelinka, I.: Symbolic regression – an overview, `http://www.mafy.lut.fi/EcmiNL/older/ecmi35/node70.html`
6. Zelinka, I., Oplatkova, Z.: Analytic programming – Comparative Study. In: CIRAS 2003: The second International Conference on Computional Intelligence, Robotics and Autonomous Systems (2003) ISSN 0219-6131
7. Lampinen, J., Zelinka, I.: New Ideas in Optimization – Mechanical Engineering Design Optimization by Differential Evolution, vol. 1, 20 p. McGraw-Hill, London (1999) ISBN 007-709506-5
8. Zelinka I.: Artificial Intelligence in The Problems of Global Optimization (in Czech). BEN, 190p. (2002) ISBN 80-7300-069-5

9. Thizzy: Be stars (2008), http://www.shelyak.com/contenu.php?id_contenu=30&id_dossier=24
10. de Veronese, L.P., Krohling, R.A.: Differential evolution algorithm on the GPU with C-CUDA. In: IEEE Congress on Evolutionary Computation, CEC (2010)
11. Kralj, P.: Differential Evolution with parallelised objective functions using CUDA (2013), http://www.codehunter.eu/media/Kralj_Differential_Evolution_with_parallelised_objective_functions_using_CUDA.pdf
12. Kromer, P., Platos, J., Snasel, V., Abraham, A.: Many-threaded implementation of differential evolution for the CUDA platform. In: Krasnogor, N. (ed.) Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO 2011), pp. 1595–1602. ACM, New York (2011), http://doi.acm.org/10.1145/2001576.2001791, doi:10.1145/2001576.2001791
13. Farber, R.: CUDA Application Design and Development, 1st edn. Morgan Kaufmann Publishers Inc., San Francisco (2011)
14. Breiman, L.: Random Forests. Machine Learning 45, 5–32 (2001)
15. Breiman, L.: Bagging predictors. Machine Learning 24, 2:123–2:140 (1996), http://dx.doi.org/10.1023/A:1018054314350, doi:10.1023/A:1018054314350
16. Caruana, R., Niculescu-Mizil, A.: An Empirical Comparison of Supervised Learning Algorithms. In: Proceedings of the 23rd International Conference on Machine Learning, pp. 161–168. ACM, New York (2006), http://doi.acm.org/10.1145/1143844.1143865, doi:10.1145/1143844.1143865
17. Ho, T.K.: The Random Subspace Method for Constructing Decision Forests. IEEE Trans. Pattern Anal. 20(8), 832–844 (1998), http://dx.doi.org/10.1109/34.709601, doi:10.1109/34.709601
18. Ben-Haim, Y., Tom-Tov, E.: A Streaming Parallel Decision Tree Algorithm. J. Mach. Learn. Res. 11, 849–872 (2010)
19. Li, B., Chen, X., Li, M.J., Huang, J.Z., Feng, S.: Scalable random forests for massive data. In: Tan, P.-N., Chawla, S., Ho, C.K., Bailey, J. (eds.) PAKDD 2012, Part I. LNCS, vol. 7301, pp. 135–146. Springer, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-30217-6_12