POLITECHNIKA BIAŁOSTOCKA

INFORMATYKA

zeszyty naukowe

ISSN 1644-0331

I

4
2009

Zeszyty Naukowe Politechniki Białostockiej

INFORMATYKA

Zeszyt 4

Oficyna Wydawnicza Politechniki Białostockiej
Białystok 2009

Redaktor naukowy:
dr hab. inż. Marek Krętowski

Recenzenci:
dr Andrzej Biela
dr inż. Ireneusz Czarnowski
dr Małgorzata Ćwiklińska-Jurkowska
prof. dr hab. inż. Marian Bolesław Gorzałczany
dr hab. Stanisław Grzegórski, prof. PL
dr hab. Ryszard Gubrynowicz
dr hab. inż. Urszula Markowska-Kaczmar
dr hab. Edward Oczeretko
prof. dr hab. inż. Wojciech Penczek
prof. dr hab. inż. Pavel Sevastsyanau
dr hab. inż. Stanisław Szczepański, prof. PG
prof. dr hab. inż. Andrzej Świerniak

Sekretarz redakcji:
mgr inż. Tomasz Łukaszuk

ISSN 1644-0331

Druk:
Oficyna Wydawnicza Politechniki Białostockiej

Nakład:100 egz.

CONTENTS
SPIS TREŚCI

ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2009
Informatyka – Zeszyt 4

Krzysztof Bandurski[1], Wojciech Kwedlo[1]

TRAINING NEURAL NETWORKS WITH A HYBRID DIFFERENTIAL EVOLUTION ALGORITHM

Abstract: A new hybrid method for feed forward neural network training, which combines differential evolution algorithm with a gradient-based approach is proposed. In the method, after each generation of differential evolution, a number of iterations of the conjugate gradient optimization algorithm is applied to each new solution created by the mutation and crossover operators. The experimental results show, that in comparison to the standard differential evolution the hybrid algorithm converges faster. Although this convergence is slower than that of classical gradient based methods, the hybrid algorithm has significantly better capability of avoiding local optima.

Keywords: neural networks, differential evolution, conjugate gradients, local minima

1. Introduction

Artificial Neural Networks with feedforward structure (ANNs) are widely used in regression, prediction, and classification. The problem of ANN training is formulated as the minimization of an error function in the space of connection weights. Typical ANN training methods e.g. backpropagation and conjugate gradient algorithms are based on gradient descent. The most advanced of them [4,12] are capable of fast convergence. However, like all *local search* methods, they are incapable of escaping from a local minimum of the error function. This property makes the final value of the error function very sensitive to initial conditions of training.

In recent years *global search* methods have received a lot of attention. Examples of such methods include evolutionary algorithms (EAs) [11] and simulated annealing [1]. EAs are stochastic search techniques inspired by the process of biological evolution. Unlike gradient descent methods they simultaneously process a *population* of problem solutions, which gives them the ability to escape from local optima. However this ability comes at the expense of very high computational complexity. This problem is especially important in neural network training where evaluation of each solution requires reading the whole learning set. A possible method for alleviating

[1] Faculty of Computer Science, Bialystok Technical University, Białystok

this drawback is construction of a hybrid algorithm which incorporates the gradient descent into the process of the evolutionary search.

The hybridization of EAs with gradient based methods can be achieved in a number of ways. In one approach, employed e.g. in the commercial DTREG package ([14]), an EA is used to locate a promising region of the weight space. Next, a gradient descent method is used to fine-tune the best solution (or all the solutions from the population) obtained by the EA. A version of this method, in which Levenberg-Marquardt algorithm is employed to refine a solution obtained by the DE was proposed in [16].

In an alternative approach, referred to in [13] as *Lamarckian* and applied for neural network training in [5], a gradient descent procedure is incorporated into an EA as a new search operator. This operator is applied to the population members in each EA iteration, in addition to standard operators such as mutation and crossover. Each application of the operator usually involves more than one iteration of a gradient descent method.

This paper is motivated by the work of Ilonen et al. presented in [9], in which they applied Differential Evolution (DE) proposed in [15] to train the weights of neural networks. They concluded that although the algorithm can converge to a global minimum, the computation time required to achive that goal can be intolerable. In an attempt to speed up the convergence rate of DE we followed the Lamarckian approach and combined it with the Conjugate Gradients ([4]) algorithm.

2. Artificial Neural Networks

A single node (artificial neuron) of an ANN receives a vector of input signals **x**, augmented by the *bias* signal which is always equal to one. The node then computes the dot-product of this input vector and the vector of weights **w** to obtain its *activation*. The output signal y emitted by the node is a usually nonlinear function of the activation, referred to as the *transfer function* or the *activation function* and denoted here as $f(net)$. It may be a simple threshold function, though other functions, like standard sigmoid or hyperbolic tangent, are often chosen for their specific properties, e.g. differentiability. The above operations can be written down as:

$$y = f(net) = f(\mathbf{x} \cdot \mathbf{w}), \tag{1}$$

where *net* is the node's activation, **x** represents the vector of input values including bias and **w** stands for the vector of weights.

In a multilayer feed forward ANN nodes are arranged in layers. A single network of this type consists of the *input layer*, which does not perform any computations and

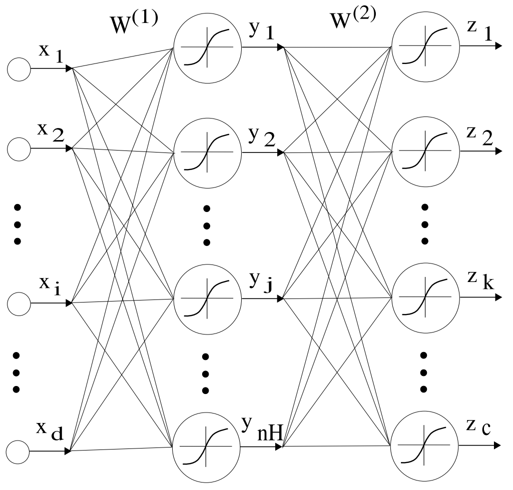


Fig. 1. A feed-forward ANN with two processing layers

only emits output signals, an arbitrary number of *processing layers*, i.e. one or more *hidden layers* and one *output layer*. Each node uses Equation (1) to compute its output signal which is then transmitted to one input of each node in the following layer. An example of an ANN with two processing layers (e.g. one hidden layer and one output layer) is presented in Fig.1. The outputs of the nodes that form the l-th processing layer in such a network can be calculated as follows:

$$\mathbf{y}_l = \mathbf{f_l}(\mathbf{x}_l\mathbf{W}_l), \tag{2}$$

where $\mathbf{x}_l = [1, y_{l-1,1} \dots y_{l-1,n} \dots y_{l-1,n_{l-1}}]$ is a vector consisting of n_{l-1} outputs signals emitted by the previous layer $l-1$ augmented with the bias signal equal to 1, $\mathbf{y}_l = [y_{l,1} \dots y_{l,n} \dots y_{l,n_l}]$ is the vector of output values yielded by the l-th layer, $\mathbf{f_l}$ is the element-by-element vector activation function used in the l-th layer and $\mathbf{W}_l$ is the matrix of weights assigned to the neurons in the l-th layer, in which a single n-th column contains all the weights of the n-th neuron, $n = 1 \dots n_l$. When l is the number of the output layer, equation (2) yields the final output vector $\mathbf{z}$.

Before an ANN can be used for prediction it must first undergo *training*. This is usually done using a *training set* T consisting of m pairs (training samples): $T = \{(\mathbf{x}_1, \mathbf{t}_1), (\mathbf{x}_2, \mathbf{t}_2), \dots, (\mathbf{x}_i, \mathbf{t}_i), \dots, (\mathbf{x}_m, \mathbf{t}_m)\}$, where $\mathbf{x}_i$ is a vector of d input values and $\mathbf{t}_i$ is a vector of c desired target values corresponding to the i-th input vector. For each input vector $\mathbf{x}_i$ supplied to the input layer the neural network yields an output vector $\mathbf{z}_i$, which is compared against the desired target vector $\mathbf{t}_i$ using a chosen *error function E*. A commonly used error function is the sum of squared errors:

$$SSE(T,W) = \sum_{i=1}^{m} \sum_{k=1}^{c} (t_{ik} - z_{ik})^2, \tag{3}$$

where W is the set of all weights in the network, t_{ik} denotes the k-th value in the i-th target vector and z_{ik} denotes the k-th output of the network produced in response to the i-th input vector. The *training error* calculated using Equation (3) is then reduced by adjusting the weights. We may thus formulate the problem of learning as the problem of minimizing the error function in the space of weights.

3. Differential Evolution

In this section the most popular *DE/rand/1/bin* differential evolution method is presented. For a more detailed description the reader is referred to [15].

Like all evolutionary algorithms, differential evolution maintains a population $U = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_s\}$ of s solutions to the optimization problem. Usually each solution takes the form of a D-dimensional real-valued vector, i.e. $\mathbf{u}_i \in \mathcal{R}^D$. At the beginning all members of the population are initialized randomly. The algorithm advances in *generations*. Each generation involves three consecutive phases: *reproduction* (creation of a temporary population), computing of the objective function (called the *fitness* in the EA terminology) for each temporary population member, and *selection*.

Reproduction in differential evolution creates a temporary population $V = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_s\}$ of *trial vectors*. For each solution $\mathbf{u}_i$ a corresponding trial vector $\mathbf{v}_i$ is created. Each element v_{ij} (where $j = 1 \ldots D$) of the trial vector $\mathbf{v}_i$ is generated as:

$$v_{ij} = \begin{cases} u_{aj} + F * (u_{bj} - u_{cj}) & \text{if } \texttt{rnd()} < CR \\ u_{ij} & \text{otherwise} \end{cases}. \tag{4}$$

In the above expression $F \in [0,2]$ is a user supplied parameter called the *mutation coefficient*. $a, b, c \in 1, \ldots, s$ are randomly selected in such a way, that $a \neq b \neq c \neq i$. `rnd()` denotes a random number from the uniform distribution on $[0,1)$, which is generated independently for each j. $CR \in [0,1]$ is another user supplied parameter called the *crossover factor*. The parameters F and CR influence the convergence speed and robustness of the optimization process. The choice of their optimal values is application dependent [15]. To alleviate the problem of finding optimal F and CR values we turned to [3] where Brest et al. proposed a self-adaptation scheme for these parameters. The values of F and CR are stored with each individual $\mathbf{u}_i$. Before they are used to generate a candidate solution $\mathbf{v}_i$ they are changed as follows:

$$F_{i,G+1} = \begin{cases} F_l + \texttt{rnd()} \; * F_u & \text{if } \texttt{rnd()} \; < \tau_1 \\ F_{i,G} & \text{otherwise} \end{cases}. \tag{5}$$

$$CR_{i,G+1} = \begin{cases} \texttt{rnd()} & \text{if } \texttt{rnd()} \; < \tau_2 \\ CR_{i,G} & \text{otherwise} \end{cases}. \tag{6}$$

where $\tau_1 = \tau_2 = 0.1$, $F_l = 0.1$ and $F_u = 0.9$, whereas $F_{i,G}$ and $CR_{i,G}$ denote the F and CR values assigned to the i-th individual in the G-th generation. The new values $F_{i,G+1}$ and $CR_{i,G+1}$ are stored with the candidate solution which may replace the original one.

The remaining two phases of a single DE generation are the computation of fitness for all members of the trial population V and the selection. The selection in differential evolution is very simple. The fitness of each trial solution $\mathbf{v}_i$ is compared to the fitness of the corresponding original solution $\mathbf{u}_i$. The trial vector replaces the original in U if its fitness is better. Otherwise the trial vector is discarded.

To apply DE to ANN training [9] the weights of all neurons are stored in a real-valued solution vector $\mathbf{u}$. The algorithm is used to minimize the sum of squared errors (SSE) or a similar criterion function. The evaluation of this function requires iterating through all elements of the training set T and summing all the partial results (squared errors in the case of SSE) obtained for all the elements of T. In the terminology of gradient-based methods, a similar approach, in which the weights are updated after the presentation of all the elements of T to the network is called *the batch training protocol* [6].

4. Conjugate Gradient Descent

The conjugate gradient algorithm (CG) was originally proposed in [8] and applied to minimize of n-dimensional functions in [7]. In [4] it was used for neural network learning as a replacement for the classical backpropagation algorithm (BP). In classical BP the vector of weights being the current estimate of the desired minimum is updated in each step by shifting it along the gradient, but in the opposite direction, according to the following formula:

$$W^{(k+1)} = W^{(k)} + \eta[-\bigtriangledown E(W^{(k)})], \tag{7}$$

where $W^{(k)}$ is the set of all m weights in the network (including biases), η is an arbitrarily chosen learning coefficient and $\bigtriangledown E(W^{(k)})$ is the gradient of E in $W^{(k)}$. In the CG algorithm, however, the gradient is used only to determine the first error descent direction, whereas each subsequent direction is chosen to be *conjugate* with all the previous ones, i.e. one along which the gradient changes only its magnitude, and not direction (in practical applications, though, the algorithm is restarted every m iterations). Another feature that differs the CG algorithm from BP is that it employs a line search algorithm in order to find a "sufficient" approximation of a minimum of the error function along a given direction. The CG algorithm works as follows:

Let $W^{(0)}$ be the initial estimate of the minimum W^* of $E(W)$, m be the size of W, $k = 0$.

[step 0]: if k mod $m == 0$ then

$$D^{(k)} = - \bigtriangledown E(W^{(k)}) \tag{8}$$

else

$$D^{(k)} = -\bigtriangledown E(W^{(k)}) + \beta_k D^{(k-1)}, \tag{9}$$

where β_k is the coefficient governing the proportion of the previous search directions in the current one. It may be one of several expressions. In our work we chose the one suggested Polak-Ribiere, which, according to [6], is more robust in non-quadratic error functions:

$$\beta_k = \frac{[\bigtriangledown E(W^{(k)})]^T [\bigtriangledown E(W^k) - \bigtriangledown E(W^{(k-1)})]}{[\bigtriangledown E(W^{(k-1)})]^T \bigtriangledown E(W^{(k-1)})} \tag{10}$$

[step 1]: Perform a line search starting at $W^{(k)}$ along direction $D^{(k)}$ to determine a step length α_k that will sufficiently approximate the minimum of the single variable function given by:

$$F(\alpha) = E(W^{(k)} + \alpha D^{(k)}) \tag{11}$$

[step 2]: Update the estimate of the minimum of $E(W)$:

$$W^{(k+1)} = W^{(k)} + \alpha_k D^{(k)} \tag{12}$$

[step 3]: $k = k+1$, go to *[step 0]*

The above procedure is repeated until a chosen termination criterion is met. It is clear that the selected line search algorithm has a significant impact on the overall performance of the algorithm presented above. We used an algorithm developed by

Charalambous, which is based on cubic interpolation. For a detailed description of this algorithm the reader is referred to [4].

5. Hybridization

As indicated in the introduction, our method of combining DE and the CG algorithm consisted in applying the latter to each candidate solution $\mathbf{v}_i$ obtained according to (4) before the computation of its fitness. The number of CG iterations is set by the user and remains constant throughout the entire experiment. By “fine-tuning” each candidate before comparing it with its predecessor we were hoping to speed up the convergence rate of DE.

6. Experiments

We tested our hybrid algorithm on 3 artificial problems and 1 real-life dataset that are described in the following paragraphs. Two versions of each of the artificial problems were tackled, each differing in the size of a single training sample. The convergence properties of our method were compared against the results yielded by self-adaptive DE with no local optimization, the Polak-Ribiere variant of Conjugate Gradient. As the computational complexity of one epoch is different in each of these algorithms, we decided to follow [9] and present our results on timescales. Each algorithm was run 30 times for each dataset. The weights of each neuron were initialized with random values from the range $\langle -\frac{1}{n_{in}}, \frac{1}{n_{in}} \rangle$, where n_{in} is the number of inputs of the neuron, including bias. Each neuron uses standard sigmoid as the activation function, whereas the error is measured by SSE. The population size in DE was set to 32. The experiments were run under Linux 2.6 on machines fitted with 64-bit Xeon 3.2GHz CPUs (2MB L2 cache) and 2GB of RAM.

6.1 Synthetic datasets

The bit parity problem. Two networks were examined: one consisting of 6 input nodes, 6 hidden nodes and 1 output node (6-6-1) and one consisting of 12 input nodes, 12 hidden nodes and 1 output node (12-12-1). The output should be set to 1 if the number of 1s in the input vector is even. The training sets consisted of 2^6 and 2^{12} samples, respectively

The encoder-decoder problem. One network consisted of 10 input nodes, 5 hidden nodes and 10 output nodes (loose encoder), whereas the other consisted of 64 input nodes, 5 hidden nodes and 64 output nodes (tight encoder). The task was to recreate the unary represention of a number presented in the input layer.

The bit counting problem. Two networks, (5-12-6) and (10-16-11) were trained to generate a unary representation of the number of bits that are set in the input vector.

6.2 Real-life dataset

The real-life dataset that we used was the *optdigits* database available in the UCI Machine Learning Repository [2]. It consists of preprocessed, normalized bitmaps of handwritten digits contributed by a total of 43 people. Each sample consists of 64 input values, representing a matrix of 8x8 where each element is an integer in the range 0..16, and 1 output value in the range of 0..9 representing the digit. The entire training set consists of 3823 samples. We modified the dataset and made each training sample contain a unary representation of the relevant digit, consisting of 10 binary values. The network that we trained consisted of 64 input nodes, 20 hidden nodes and 10 output nodes.

6.3 Results

Results are presented in tables and graphs. In the case of the synthetic datasets, each table consist of three columns. The first one contains the names of algorithms that were compared in our study: cgpr stands for Conjugate Gradients described in section 4., DE denotes the Differential Evolution algorithm presented in section 3., whereas DE-cgpr-z*xxx* corresponds to Differential Evolution augmented with the conjugate gradient algorithm as described in section 5., where *xxx* denotes the number of iterations of Conjugate Gradients that were applied to each individual before the selection phase. The two other columns contain the results obtained for each variant of the dataset (the respective network configuration and the duration of each run are given in the header of each column). These results were averaged over 30 independent runs and include: the sum of squared errors divided by 2 (SSE), the standard deviation (σ), the number of forward passes of the entire training set through the network per second (fp/s) and the number of backward passes of the error (bp/s) and finally the mean number of DE generations (gen).

Below each table there are two graphs, each presenting 6 mean error curves reflecting the progress of the tested algorithms.

The results obtained for the real-life dataset are presented in a similar manner, with the difference being that only one network configuration was used.

Table 1. Results obtained for the bit parity problems

algorithm	(6-6-1) - 5 min						(12-12-1) - 4 h				
	SSE	σ	fp/s	bp/s	gen		SSE	σ	fp/s	bp/s	gen
cgpr	0.457	1.423	11398.3	11398.3	n/a		512.00	0.00	75.7	75.7	n/a
DE	2.609	0.593	18826.2	0.0	176494.7		321.80	175.97	133.5	0.0	60063.3
DE-cgpr-z008	0.285	0.229	11585.3	11585.2	2591.3		108.20	35.27	76.0	76.0	884.2
DE-cgpr-z016	0.044	0.113	11578.3	11578.2	1324.6		56.39	20.10	76.0	76.0	439.7
DE-cgpr-z032	0.000	0.000	11595.8	11595.7	671.1		26.62	13.91	76.0	76.0	217.0
DE-cgpr-z064	0.000	0.000	11605.4	11605.3	342.1		26.56	10.63	76.0	76.0	96.5

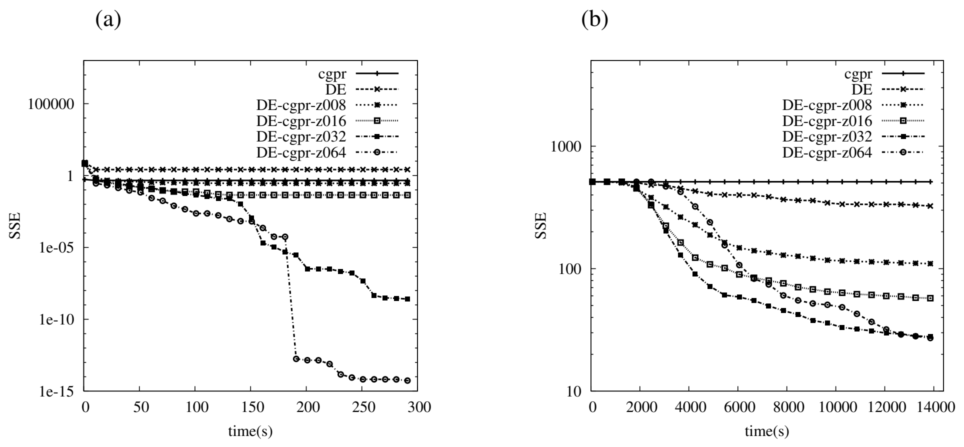


Fig. 2. SSE curves for the 6-bit (a) and 12-bit (b) parity problems

7. Conclusions

The results presented above demonstrate that the hybrid algorithm converges significantly faster than the original version of DE used in [9], provided that the number of iterations of the local optimization algorithm is sufficient. In two cases (namely 64-5-64 encoder-decoder, DE-cgpr-z008 and 10-bit counting, DE-cgpr-z008) the hybrid solution achieved a greater error value than the one achieved by the unmodified version of DE, but as the number of iterations of conjugate gradient descent increased, the hybrid version proved to be superior. This supports the findings presented by Cortez in [5], who advocated the use of the Lamarckian approach.

Our solution's capability to escape local minima is visible the most in figures 2 (a), 2 (b), 3 (a), 5, which also demonstrate its superiority in comparison to the

Table 2. Results obtained for the encoder-decoder problems

algorithm	(10-5-10) - 5 min						(64-5-64) - 4 h				
	SSE	σ	fp/s	bp/s	gen		SSE	σ	fp/s	bp/s	gen
cgpr	0.017	0.090	35535.8	35535.8	n/a		1.85	6.59	1048.7	1048.7	n/a
DE	0.075	0.169	44603.3	0.0	418154.8		15.61	0.88	1546.9	0.0	696124.0
DE-cgpr-z008	0.000	0.000	35569.4	35569.3	7141.7		17.99	1.24	1044.3	1044.3	12602.6
DE-cgpr-z016	0.000	0.000	35615.8	35615.7	3785.0		13.15	2.29	1044.9	1044.9	6463.8
DE-cgpr-z032	0.000	0.000	35662.8	35662.7	1949.1		7.72	1.60	1045.2	1045.2	3056.6
DE-cgpr-z064	0.000	0.000	35696.9	35696.8	1003.9		1.66	1.00	1045.2	1045.2	1045.5

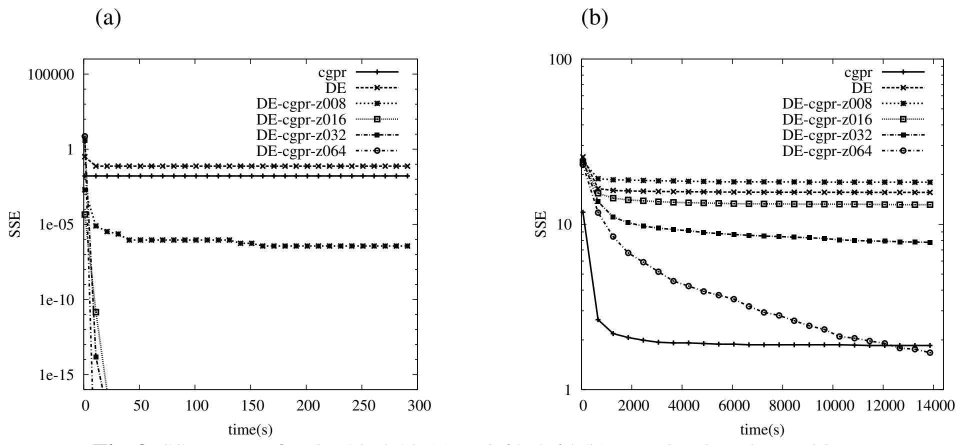


Fig. 3. SSE curves for the 10-5-10 (a) and 64-5-65 (b) encoder-decoder problems

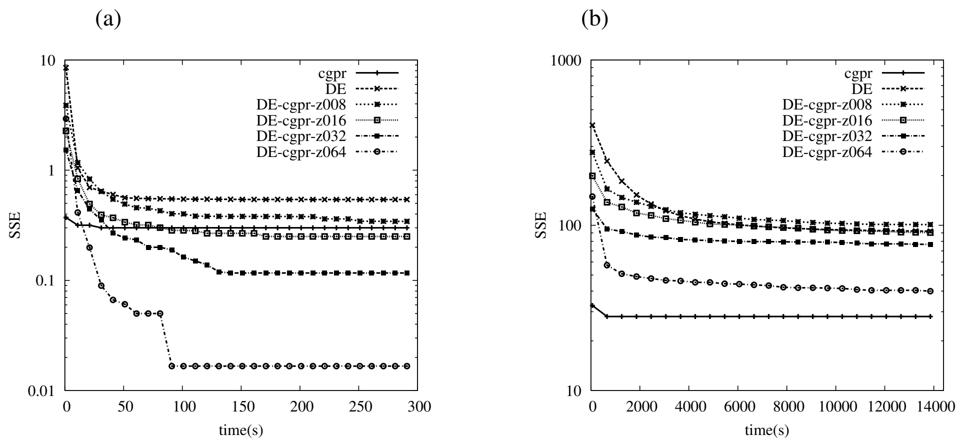


Fig. 4. SSE curves for the 5 bit (a) and 10-bit (b) counting problems

Table 3. Results obtained for the bit counting problems

algorithm	(5-12-6) - 5 min						(10-16-11) - 4 h				
	SSE	σ	fp/s	bp/s	gen		SSE	σ	fp/s	bp/s	gen
cgpr	0.300	0.355	9996.4	9996.4	n/a		28.07	34.67	152.4	152.4	n/a
DE	0.541	0.262	14971.5	0.0	140356.9		91.93	10.26	269.6	0.0	121326.9
DE-cgpr-z008	0.343	0.295	9893.8	9893.7	2255.8		100.73	12.47	153.8	153.8	1877.8
DE-cgpr-z016	0.250	0.250	9904.6	9904.5	1181.0		89.89	10.36	153.8	153.8	909.7
DE-cgpr-z032	0.117	0.211	9912.5	9912.4	579.0		76.71	6.66	153.8	153.8	434.0
DE-cgpr-z064	0.017	0.090	9917.7	9917.6	278.9		39.94	8.00	153.8	153.8	216.7

Table 4. Results obtained for the optdigits dataset

	(64-20-10) - 16 h				
algorithm	SSE	σ	fp/s	bp/s	gen
cgpr	54.75	129.9	10.112	10.112	n/a
DE	36.55	3.56	17.3	0.000	31121.0
DE-cgpr-z008	8.15	5.51	10.0	10.0	510.9
DE-cgpr-z016	4.33	1.90	10.0	10.0	261.9
DE-cgpr-z032	1.34	0.64	10.0	10.0	130.6
DE-cgpr-z064	0.35	0.40	10.0	10.0	64.0

conjugate gradient method. Figures 3 (b) and 4 (b) may suggest, however, that for some problems the hybrid algorithm performs much worse. In these cases it is worth to pay attention to the standard deviation values computed on the basis of all 30 runs performed for each dataset. They are much higher for the conjugate gradient method, which implies that under pre-defined time constraints the hybrid algorithm could require a smaller number of longer runs to yield results at an acceptable level of certainty.

Another advantage of our algorithm is its potential parallelizability. In [10] the authors proposed a method for parallelizing the original version of DE that is based on the decomposition of not only the dataset, but also the population of solutions. The algorithm presented here can be parallelized in a similar manner, with some additional mechanisms to support backpropagation. This subject is currently being investigated and will be presented in a following paper.

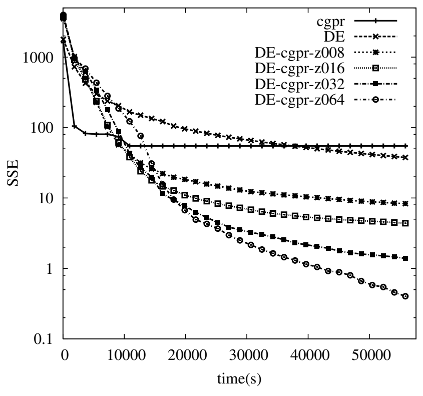


Fig. 5. SSE curves for the optdigits problem

References

[1] E. H. L. Aarst and J. Korst: Simulated Annealing and Boltzmann Machines, John Wiley, 1989.

[2] C. Blake, E. Keogh, and C. J. Merz: UCI repository of machine learning databases, University of California, Dept. of Computer Science, http://www.ics.uci.edu/∼mlearn/MLRepository.html, 1998.

[3] J. Brest, S.Greiner, B. Boskovic, M. Mernik, and V.Zumer: Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems, IEEE Transactions on Evolutionary Computation, 10(6):646–657, 2006.

[4] C. Charalambous: Conjugate gradient algorithm for efficient training of artificial neural networks, Circuits, Devices and Systems, IEE Proceedings G, 139:301–310, 1992.

[5] Paulo Cortez, Miguel Rocha, and Jos Neves: A lamarckian approach for neural network training Neural Processing Letters, 15:105–116, 2002.

[6] R. O. Duda, P. E. Hart, and D. G. Stork: Pattern Classification, John Wiley and Sons, 2001.

[7] R. Fletcher and C. M. Reeves: Function minimization by conjugate gradients, The Computer Journal, 7:149–154, 1964.

[8] M. R. Hestenes and E. Stiefel: Methods of conjugate gradients for solving linear systems, Journal of Research of the National Bureau of Standards, 49:409–436, 1952.

[9] J. Ilonen, J. K. Kamarainen, and J. Lampinen: Differential evolution training algorithm for feed-forward neural networks, Neural Processing Letters, 17:93–105, 2003.

[10] W. Kwedlo and K. Bandurski: A parallel differential evolution algorithm for neural network training, In Parallel Computing in Electrical Engineering, 2006. PARELEC 2006. International Symposium on, pages 319–324. IEEE Computer Society Press, 2006.

[11] Z. Michalewicz: Genetic Algorithms + Data Structures = Evolution Programs, Springer Verlag, 1996.

[12] M. F. Møller: A scaled conjugate gradient algorithm for fast supervised learning, Neural Networks, 6(4):525–533, 1993.

[13] Brian J. Ross: A lamarckian evolution strategy for genetic algorithms, In Lance D. Chambers, editor, Practical Handbook of Genetic Algorithms: Complex Coding Systems, volume 3, pages 1–16. CRC Press, Boca Raton, Florida, 1999.

[14] Phillip H. Sherrod: Dtreg - predictive modeling software, 2008.

[15] R. Storn and K. Price: Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization, 11:341–359, 1997.

[16] B. Subudhi and D. Jena: Differential evolution and Levenberg Marquardt trained neural network scheme for nonlinear system identification, Neural Processing Letters, 27(3):285–296, 2008.

UCZENIE SIECI NEURONOWYCH HYBRYDOWYM ALGORYTMEM OPARTYM NA DIFFERENTIAL EVOLUTION

Streszczenie: W artykule przedstawiono nową, hybrydową metodę uczenia sieci neuronowych, łączącą w sobie algorytm Differential Evolution z podejściem gradientowym. W nowej metodzie po każdej generacji algorytmu Differential Evolution każde nowe rozwiązanie, powstałe w wyniku działania operatorów krzyżowania i mutacji, poddawane jest kilku iteracjom algorytmu optymalizacji wykorzystującego metodę gradientów sprzężonych. Wyniki eksperymentów wskazują, że nowy, hybrydowy algorytm ma szybszą zbieżność niż standardowy algorytm Differential Evolution. Mimo, iż zbieżność ta jest wolniejsza, niż w przypadku klasycznych metod gradientowych, algorytm hybrydowy potrafi znacznie lepiej unikać minimów lokalnych.

Słowa kluczowe: sieci neuronowe, differential evolution, gradienty sprzężone, minima lokalne

ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2009
Informatyka – Zeszyt 4

Maciej Brzozowski[1], Vyacheslav Yarmolik[1]

OBFUSCATION QUALITY IN HARDWARE DESIGNS

Abstract: Software is more and more frequently distributed in form of source code. Unprotected code is easy to alter and build in others projects. The answer for such attacks is obfuscation. Transformation of software source code which preserves its semantical functionality but analizability is made very dificult. We focus in our research on Very High Speed Integrated Circuits Hardware Description Language (VHDL). In previous articles we presented transformats assimilated from other high level languages for needs of hardware designs and we showed a set of new transformants which do not appear in different languages than VHDL. The next step we have to do is to establish a set of criterias through which we can rate quality of analyzed transformats. In article we present current bacground of software quality measure and we rate their usage for protecting intellectual property on digital designs. After that we propose a new way of obfuscation quality measure taking into account requirements of hardware designs.

Keywords: Obfuscation, Very High Speed Integrated Circuits Hardware Description Language (VHDL), Intelectual Property Protection

1. Introduction

Hackers' activity shows that it should secure not only software before stealing but also a hardware projects. Source code is stolen and built in other projects. Many of the security tools that can be used to protect the software were created like watermarks, fingerprints and obfuscation (Figure 1).

The purpose of watermarking [9,14,10] is to insert a mark (owner copyright notice) to the program that is invisible and difficult to remove. With the watermark, it is possible to prove the owner rights to the code or project.

Next technique for protecting projects is fingerprinting [7,6]. Fingerprinting is similar to watermarking, except a different (unique) secret message is embedded in every distributed cover message. A typical fingerprint includes a seller, product, and customer identification numbers. This allow to detect when and where theft has occurred and to trace the copyright offender.

[1] Faculty of Computer Science, Bialystok Technical University, Białystok

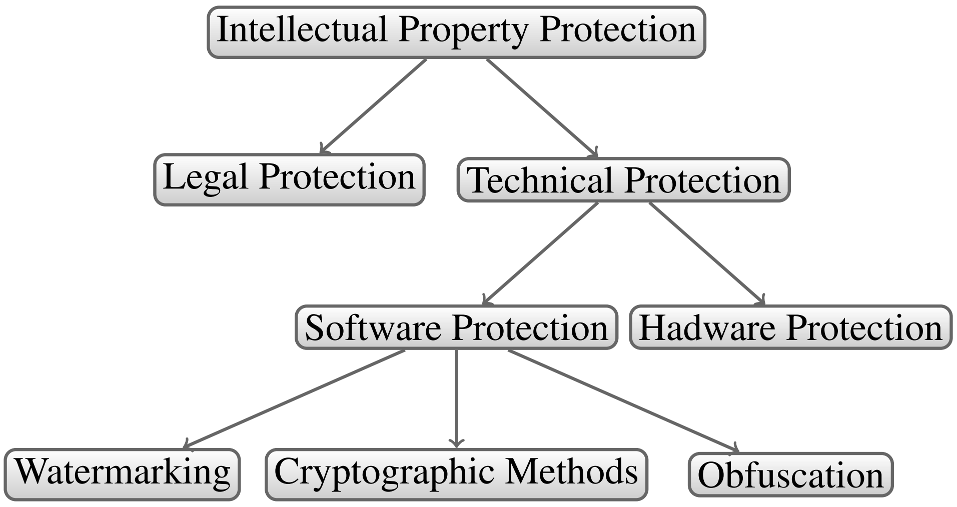


Fig. 1. Ways of intelectual property protection.

The last one technique is obfuscation [5] - one of the most modern and effective software protection technique. Obfuscation alters the code so that it is very difficult to read and understand. This type of protection does not protect against illegal copying and execution. The aim of obfuscation is prevention against stealing parts of source code (analysing and rebuilding in) and on the reverse engineering.

Subject of obfuscation is so young so it is hard to tell about its history. The first articles date from the second half of nineties. The precursor of subject was Christian Collberg. He wrote the first publication connected with obfuscation [5]. In several articles, which coauthor he is, Collberg presents techniques of code obfuscation and divide them into four groups by the target application - layout, data, control and preventive transformations. A lot of articles were written but no one concerned obfuscation of VHDL code. It is very easy to change code of other language like JAVA or C# because size and execute time of transformed program is not so significant. In VHDL these metrics are the most important for a software engineer.

The purpose of our work is to study obfuscation techniques and choose most suitable of them to protecting intellectual property rights on VHDL source code. We should remember that this language in comparison with other most frequently used high level languages is, in his building, unique. VHDL language makes uses of parallel processing and a lot of obfuscation techniques which can not be used in such languages like C# or Java are very useful and relatively cheep.

2. Definition of obfuscation

For people who are not familiar with the subject of obfuscation, it is intentional action conducting to modify the software code in such way that it becomes difficult to understand. Gaël Hachez in [9] changed definition of obfuscation created by Collberg in [5] and also used in [8] to:

Definition 1 (Obfuscating Transformation) $P \xrightarrow{\tau} P'$ be a transformation of a source program P into a target program P'. $P \xrightarrow{\tau} P'$ is an obfuscating transformation, if P and P' have the same observable behavior. More precisely, in order for $P \xrightarrow{\tau} P'$ to be a legal obfuscating transformation the following conditions must hold:

a) If P fails to terminate or terminates with an error condition, then P' fails to terminate or terminates with an error.
b) Otherwise, P' must terminate and produce the same output as P.

We should remember, that every program P' will be possible to reconstruct to P'', which will have very similar structure to P. Obviously, such operation will absorb much time and costs, but always it will be possible [3]. Therefore problem of obfuscation is not a protection before decompilation of program, but makes it very difficult.

Listing 1.1. Winner of 1st International Obfuscated C Code Contest in category Dishonorable mention - 1984

```
int i;main(){for(;i["]<i;++i){--i;}"];read('-'-'-',i+++"hell\
o,␣world!\n",'/'/'/'));}read(j,i,p){write(j/p+p,i---j,i/i);}
```

In Listing 1.1 is showed program source code anonimous.c - winner of 1st International Obfuscated C Code Contest in category Dishonorable mention. Obfuscated example shows how complicated analysis of protected code may be. For people who would like to take analysis of code and check its correctness we place base code (Listing 1.2).

Listing 1.2. Hello word. Code before obfuscation.

```
#include<stdio.h>
int main(void){
    printf("Hello␣World!\n");
    return 0;}
```

3. Quality characteristics - theoretical background

Before we attempt to describe obfuscation transforms, we need to be able to evaluate their quality. In this section we describe theoretical background of quality code measurement. We will also examine different new approaches to this problem.

3.1 International Standard ISO/IEC-9126

In 1991 the International Standards Organization introduced standard ISO/IEC-9126 [1], whose task was to define the standards and requirements for the description of the software. In subsequent years, the standard has been extended by four characteristics:

- ISO/IEC 9126-1:2001 Quality model
- ISO/IEC 9126-2:2003 External metrics
- ISO/IEC 9126-3:2003 Internal metrics
- ISO/IEC 9126-4:2004 Quality in use metrics

It is currently the most widely known and wide-spread quality standard of software. The standard ISO/IEC-9126 is often confused with the characteristics ISO/IEC-9126-1:2001 [2] introduced in 2001. The model defines the following six main quality characteristics:

- Functionality - a collection of attributes that describe a set of software functions and their designated properties.
- Reliability - a collection of attributes describing the capabilities of software to maintain the specified requirements as to its stability under strictly defined conditions of work.
- Usability - a collection of attributes that describe the level of work needed to learn to use the software and the subjective assessment of the attractiveness of the software expressed by group members.
- Efficiency - a measurement of the use of system resources compared to the performance of software subject to certain conditions.
- Maintainability - a collection of attributes that describe the level of work required to bring about changes in the software.
- Portability - a collection of attributes describing the product the ability to transfer programming between other systems.

They are supported by less significant characteristics in order to increase the precision of individual metrics.

3.2 Collberg's quality measures

In 1997 in the technical report [5] Christian Collberg with Clark Thomborson and Douglas Low defined the concept of obfuscation and attempted to assess its quality.

They defined quality of obfuscation as combination of:

- Potency - degree of difficulty understanding obfuscated code for the analysing person. Influence on the power of transformation have: introduce new class and methods, increase predicates and nestling level of conditional, increase the highest of the inheritance tree, increase long-range variables dependencies.
- Resilience - how much time a developer has to spend to implement automatic deobfuscator, which can effectively lower the power of transform.
- Cost - metric applies to both increase the cost of the implementation of the program as well as increased demand for resources like memory or CPU time.

In the lectures [16] Collberg previously defined range of metrics expends by a descriptive metric stealth. Its estimate is dependent on the context in which it was used.

3.3 Petryk's software metrics

In [15] Siarhei Petryk with Ireneusz Mrozek and Vyacheslav N. Yarmolik proposed metric composed from two submetrics. First group based on code complexity like the number of conditional operators compared to all words in the program or average depth of branching operators. The second based on program execution time and its length. Values of metrics from first class should be maximized whereas from the second class should be minimized. They assumed that the best obfuscator optimise the program code simultaneously makes source code more confusing and harder to interpret.

4. Obfuscation quality - new approach

Presented above metrics have been developed for non hardware languages. For example size of the code does not have such signification in designing hardware systems in opposite to the software projects. We are not bound by the amount of RAM or CPU speed. Above metrics do not take into account the signal delay propagation time on critical path which is one of most significant metric in digital designs and not occur in software projects. Therefore, there is a need to develop metrics which are intended solely to evaluate the quality obfuscation transforms used in hardware projects.

First step in design quality metric is divide metrics in two groups:

- critical metrics - most important metrics like the path delay of the system and the amount of used area on a chip.
- less important metric but not meaningless like complexity and code size.

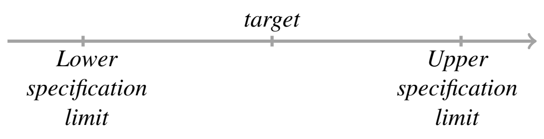


Fig. 2. Target and tolerance for metrics.

ISO/IEC-9126 [1] does not define how to measure metrics described in it. Some of quality attributes are subjective and difficult to measure. The standard describe only how the metrics should work and measure of them leaves for experts. We can not rely on metrics which estimate will be so costly. By expensive we understand the time effort and resources needed for estimate. The better expert that his time is more and more expensive.
Effective metrics satisfy the following conditions:

- performance is clearly defined in a measurable entity - the metric is quantifiable.
- exists a capable system to measure metric in the entity.

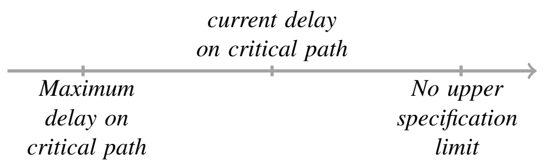

current delay on critical path
Maximum delay on critical path
No upper specification limit

Fig. 3. Target and tolerance metric for critical path delay in obfuscated design.

The second criteria that metrics must be estimated taking into account is the parameters imposed by the user. In Figure 2 is presented target value of metric witch allowable deviation where a characteristic will be still acceptable. Lower specification limit (LSL) and upper specification limit (USL) are break points where make worse performance will make design unable to work properly and improve make design extremely useful.

It is mean that user has to know metric or estimated value expected ranges for obfuscated design. Figure 3 present maximal delay on critical path propagation of protected design source code. The propagation delay reflects how fast system can operate and is usually considered as speed of the system. Minimal delay is not defined because delay lower than maximal or in special cases lower than nominal delay will be always acceptable if we consider only combinational circuits. In other cases we have to remember about triggers time specification - setup time and hold time. We should note that reach of low value of propagation delay will lead to high costs of code transformation.

The second most important metric is area on the target chip used by obfuscated design. As in the case of propagation delay user does not define the upper value of the used area because it is not necessary and any value in range below maximal would be acceptable. Design area greater than maximal would not fit in target chip. Circuit area combined with propagation delay is most important design criteria in digital systems.

The third most significant metric is cost of apply obfuscation transformation. We count the cost as effort of resources such as time of execute transformation, implementation effort and human resources needed.

$$E_1 = a_1 \frac{t_d}{t'_d} + a_2 \frac{area}{area'} + \frac{a_3}{E_c}$$

$$LSL_{t_d} = \frac{t_d}{t_{d_{max}}} \Rightarrow \frac{t_d}{t_{d_{max}}} \leqslant \frac{t_d}{t'_d} \Rightarrow t_{d_{max}} \geqslant t'_d$$

$$LSL_{area} = \frac{area}{area_{max}} \Rightarrow \frac{area}{area_{max}} \leqslant \frac{area}{area'} \Rightarrow area_{max} \geqslant area'$$

$$E_c = \{free \mapsto 1,\ cheap \mapsto 2,\ expensive \mapsto 2^2,\ very\ expensive \mapsto 2^3\}$$

t_d	- nominal value of propagation delay on critical path
$t_{d_{max}}$	- maximal value of propagation delay on critical path
t'_d	- value of propagation delay on critical path after obfuscation transformation applying
$area$	- nominal value of used area
$area_{max}$	- maximal value of used area
$area'$	- value of used area after obfuscation transformation applying
E_c	- cost of apply obfuscation transformation
a_1, a_2, a_3	- weight of the importance of metrics

The second group describes how good obfuscated code is. It is very easy to describe how correctly written code should look but it is very difficult to reflect it in metrics. The first easiest to estimate submetric is number of lines of code (LOC). Counting lines of code is not reliable in high level languages therefore quality metric should based on instruction statements to (IS). More instruction statements to line of code increase submetric value and makes analysis difficult.

Other useful metric is cyclomatic complexity defined by Thomas McCabe in [13]. The measure describe the number of linearly independent (distinct) paths through a program's flow graph. To have poor testability and maintainability we recommends that program module should exceed a cyclomatic complexity of 10. This high level of metric will fuzzy logical pattern of design.

Our quality metric lets user to choose more suitable metrics for him. User might chose for example:

- one or more from Halstead metrics [11] - metric currently less used
- Henry and Kafura's structure complexity metric [12] - increasing fan-in and fan-out will decrease readability and analyzability of project

$$E_2 = a_4 \frac{E'_{CC}}{E_{CC}} + a_5 \frac{E'_{IS/LOC}}{E_{IS/LOC}} \underbrace{+ a_6 \frac{V'_p}{V_p} + a_7 \frac{C'_P}{C_P} + \cdots}_{optional}$$

$$E_{IS/LOC} = \frac{IS}{LOC}$$

CC	- Cyclomatic Complexity
IS	- Instruction Statements
LOC	- Lines Of Code
V	- Halstead Volume
C_P	- Henry and Kafura's structure complexity
$a_4, \ldots$	- weight of the importance of metrics

On the basis of previous parts of metrics we achieved our goal:

$$E = E_1 + E_2$$

E_1 is a group of the most important metrics for digital designs and E_2 represents complexity of obfuscated code.

5. Estimation quality of obfuscating transformation

Layout obfuscation [5] relies on changing (removing) information in the source code that does not effect operation of program. It is often called free because does not influence on size of program neither on its speed of operation after transformation. There is one-way transformation because once deleted information can not be restored. Amount of helpful information decreases for analysing person. This technique contains removing comments, descriptions and changing names of variables which suggest what is it for. For scrambling identifiers does not change propagation delay on critical path nor project area on the target chip.

$$E_{sc} = a_1 + a_2 + a_3 + a_4 + a_5$$

$$\frac{t_d}{t'_d} = \frac{area}{area'} = \frac{E'_{CC}}{E_{CC}} = \frac{E'_{IS/LOC}}{E_{IS/LOC}} = 1$$

$$E_c = free = 1$$

It is easy to see that layout obfuscation is one of the cheapest techniques allowed to protect intellectual property.

There is transformation which does not appear in other programming languages. In view of its specification VHDL is based on parallel processing instruction. Almost every code written in VHDL may be converted from or to sequential processing.

Listing 1.3. Conversion of sequential processing to parallel

```
process (x)
        variable tmp : std_logic;
begin
        tmp := '0';
        for i in x'range loop
                tmp := tmp xor x(i);
        end loop;
        y <= tmp;
end process;
```

$$\Downarrow$$

```
tmp <= tmp(N–2 downto 0)&'0' xor x;
y <= tmp(N−1);
```

Mentioned above transform is not difficult to realize. Listing 1.3 presets two obfuscation techniques from group of control transformation. We used technique called unrolling loops connected with conversion of sequential processing to parallel [4]. Transformation does not influence on critical part of metric.

6. Conclusion

We have developed the first hardware design obfuscation quality metering scheme. The metric takes into account the signal delay propagation time on critical path and the amount of used area by design on a target chip. Moreover with this quality tool a user can choose obfuscation transforms answering his requirements of created project.

References

[1] ISO/IEC-9126, International Standard ISO/IEC. In *Information technology: Software product evaluation: Quality characteristics and guidelines for their use*. International Standards Organisation, 1991.

[2] ISO/IEC-9126-1:2001, International Standard ISO/IEC 9126. In *Information Technology – Product Quality – Part1: Quality Model*. International Standard Organization, June 2001.

[3] Impagliazzo R. Rudich S. Sahai A. Vadhan S. Yang K. Barak B., Goldreich O.: On the (im)possibility of obfuscating programs, Lecture Notes in Computer Science, 2139:1–14, 2001.

[4] Yarmolik V. N. Brzozowski M.: Vhdl obfuscation techniques for protecting intellectual property rights on design, 5th IEEE East-West Design and Test Symposium, pages 371–375, 2007.

[5] Low D. Collberg C., Thomborson C.: A taxonomy of obfuscating transformations, Technical report, July 1997.

[6] Thomborson C. Collberg C.: The limits of software watermarking, 1998.

[7] Thomborson C. Collberg C.: Watermarking, tamper-proofing, and obfuscation – tools for software protecti on, Technical Report TR00-03, Thursday, 10 2000.

[8] Low D. Collberg C. S., Thomborson C. D.: Breaking abstractions and unstructuring data structures, In International Conference on Computer Languages, pages 28–38, 1998.

[9] Hachez G.: A comparative study of software protection tools suited for e-commerce with contributions to software watermarking and smart cards, Universite Catholique de Louvain, March 2003.

[10] Wroblewski G.: General Method of Program Code Obfuscation, PhD thesis, Wroclaw University of Technology, Institute of Engineering Cybernetics, 2002.

[11] Halstead M. H.: Elements of Software Science, North-Holland, Amsterdam, The Netherlands, 1977.

[12] Kafura D. Henry S.: Software structure metrics based on information flow, 7(5):510–518, September 1981.

[13] McCabe T. J.: A complexity measure, IEEE Trans. Software Eng., 2(4):308–320, 1976.

[14] Petitcolas F. A. P. Katzenbeisser S.: Information hiding - techniques for steganography and digital watermarking, Artech House, Norwood, 2000.

[15] Yarmolik V. N. Petryk S., Mrozek I.: Efficiency of obfuscation method based on control transformation, In ACS, pages 351–359, 2006.

[16] www.cs.arizona.edu/∼collberg/Research/Publications/index.html.

JAKOŚĆ OBFUSKACJI PROJEKTÓW SPRZĘTOWYCH

Streszczenie Obfuskacja jest techniką przekształcania kodu źródłowego oprogramowania, który zachowuje swoje działanie semantyczne, ale znacząco zostaje utrudniona jego analiza oraz zrozumienie. Swoje badania skupiliśmy na sprzętowym języku Very High Speed Integrated Circuits Hardware Description Language (VHDL). W poprzednich pracach przestawiliśmy szereg transformat zaasymiowanych z języków wysokiego poziomu na potrzeby projektów sprzętowych oraz zaproponowaliśmy nowe nie występujące w innych językach niż VHDL. Kolejnym krokiem jaki należy wykonać jest ustalenie kryteriów dzięki którym będzie można ocenić jakość analizowanych transformat. W artykule przedstawimy dotychczas używane metody oceny jakości oprogramowania oraz przeprowadzimy ich ocenę na potrzeby ochrony własności intelektualnej projektów sprzętowych. Następnym krokiem będzie przedstawienie nowego sposobu oceny jakości obfuskacji z uwzględnieniem wymagań jakie stawiają przed programistą projekty sprzętowe.

Słowa kluczowe: Obfuskacja, VHDL, ochrona własności intelektualnej

ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2009
Informatyka – Zeszyt 4

Marcin Czajkowski[1], Marek Krętowski[1]

AN EXTENSION OF TSP-FAMILY ALGORITHMS FOR MICROARRAY CLASSIFICATION

Abstract: Classification of microarray data and generation of simple and efficient decision rules may be successfully performed with Top Scoring Pair algorithms. *TSP*-family methods are based on pairwise comparisons of gene expression values. This paper presents a new method, referred as *Linked TSP* that extends previous approaches $k-TSP$ and *Weight* $k-TSP$ algorithms by linking top pairwise mRNA comparisons of gene expressions in different classes. Opposite to existing *TSP*-family classifiers, the proposed approach creates decision rules involving single genes that most frequently appeared in top scoring pairs. Motivation of this paper is to improve classification accuracy results and to extract simple, readily interpretable rules providing biological insight as to how classification is performed. Experimental validation was performed on several human microarray datasets and obtained results are promising.

Keywords: pairwise classification, decision rules, microarray, gene expression

1. Introduction

DNA chips technology has given rise to the study of functional genomics [3,14]. The entire set of genes of an organism can be microarrayed on an area not greater than 1 cm^2 and enable to analyze hundred of thousands of expression levels simultaneously in a single experiment [7]. Microarray technology make possible comparisons of gene expressions levels and computational analysis allows classification samples by their mRNA expression values.

Nowadays, DNA chips are widely used to assist diagnosis and to discriminate cancer samples from normal ones [2,6]. Extracting accurate and simple decision rules that contain marker genes is of great interest for biomedical applications. However, finding a meaningful and robust classification rule is a real challenge, since in different studies of the same cancer, diverse genes consider to be marked [16].

Typical statistical problem that often occurs with microarray analysis is dimensionality and redundancy. In particular, we are faced with the *"small N, large P problem"* [17,18] of statistical learning because the number of samples (denoted by *N*)

[1] Faculty of Computer Science, Bialystok Technical University, Poland

comparing to number of features/genes (P) remains quite small as N usually does not exceeded one or two hundreds where P is usually several thousands. This may influence the model complexity [11] and cause the classifier to overfit training data. Considering some dimensionality reduction (i.e. feature selection) seems to be reasonable as most of the genes are known to be irrelevant for classification and prediction. Applying gene selection prior classification [15] may simplify calculations, model complexity and often improve accuracy of the following classification.

Recently, many new solutions based on classification approaches including statistical learning and pattern recognition methods are applied to microarray data [19,10]. However most of them generate very complex decision rules that are very difficult or even impossible to understand and interpret. This is a trade-off between credibility and comprehensibility of the classifiers [20].

In this paper, we would like to propose an alternative approach for TSP-family classifiers. The presented solution (denoted as *Linked* TSP) may be applied to original TSP classifier [8] or its extensions: $k-TSP$ [20] and *Weight* $k-TSP$ [5]. In our research we have experimentally observed that some genes, more often to the others, appear in top pairs calculated by one of these TSP algorithms. This may suggest that some genes from the list of top pairs more accurate discriminate cancer samples from normal one. Our method is focused on finding predominatingly genes from calculated top pairs of genes. We believe that these approach will simplify decision rules without reducing classification accuracy or even improve it for some datasets.

The rest of the paper is organized as follows. In the next section TSP-family algorithms are briefly recalled. Section 3 describes proposed solution - *Linked* TSP. In section 4 the presented approach is experimentally validated on real microarray datasets. The paper is concluded in the last section and possible future works are suggested.

2. A Family of TSP Algorithms

TSP-family methods are applied according to the classical supervised learning framework. In the first step the top scoring pairs are generated from the training dataset. This process is illustrated in Fig. 1. In the second step, the obtained classifier can be applied to a new microarray sample with unknown decision class. Only selected genes called "marker genes" are analyzed and used in TSP prediction (Fig. 2). *Linked* TSP classifier uses only first step of TSP-family methods to obtain sorted (decreasingly by significance) list of gene pairs generated by these algorithms.

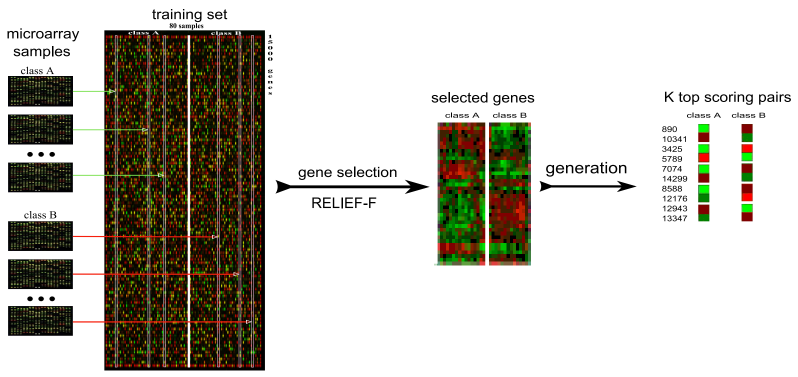


Fig. 1. Building TSP-based decision rules on the training dataset

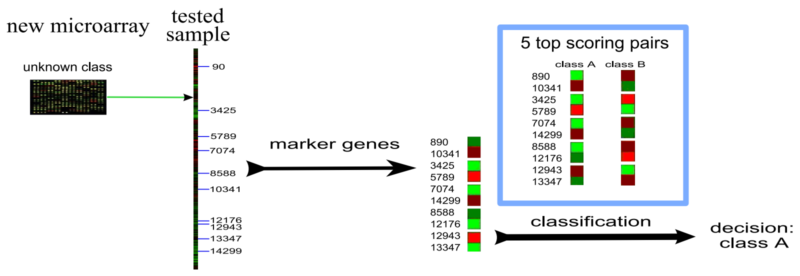


Fig. 2. Testing a new sample with the TSP classifier based on the selected genes

2.1 Top Scoring Pair

Top Scoring Pair (TSP) method was presented by Donald Geman [8] and is based on pairwise comparisons of gene expression values. Despite its simplicity comparing to other methods, classification rates for TSP are comparable or even exceeds other classifiers [8]. Discrimination between two classes depends on finding pairs of genes that achieve the highest ranking value called "score".

Consider a gene expression profile consisting of P genes and N samples participating in the training microarray dataset. These data can be represent as a $P \times N$ matrix X:

$$X = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1N} \\ x_{21} & x_{22} & \dots & x_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{P1} & x_{P2} & \dots & x_{PN} \end{pmatrix},$$

in which the expression value of i-th gene from the n-th sample is denoted by x_{ij}. Each row represents observations of a particular gene over N training samples, and each column represents a gene expression profile composed from P genes. Each profile has a true class label denoted $C_m \in C = \{C_1, \ldots, C_M\}$. For the simplicity of calculations it is assumed that there are only two classes $(M = 2)$ and profiles with indices from 1 to N_1 $(N_1 < N)$ belong to the first class (C_1) and profiles from range $\langle N_1 + 1, N \rangle$ to the second class (C_2).

TSP method focuses on gene pair matching (i, j) $(i, j \in \{1, \ldots, P\}, i \neq j)$ for which there is the highest difference in the probability p of an event $x_{in} < x_{jn}$ $(n = 1, 2, \ldots, N)$ between class C_1 and C_2. For each pair of genes (i, j) two probabilities are calculated $p_{ij}(C_1)$ and $p_{ij}(C_2)$:

$$p_{ij}(C_1) = \frac{1}{|C_1|} \sum_{n=1}^{N_1} I(x_{in} < x_{jn}) \; , \tag{1}$$

$$p_{ij}(C_2) = \frac{1}{|C_2|} \sum_{n=N_1+1}^{N} I(x_{in} < x_{jn}) \; , \tag{2}$$

where $|C_m|$ denotes a number of profiles from class C_m and $I(x_{in} < x_{jn})$ is the indicator function defined as:

$$I(x_{in} < x_{jn}) = \begin{cases} 1, & \text{if } x_{in} < x_{jn} \\ 0, & \text{if } x_{in} \geq x_{jn} \end{cases} \; . \tag{3}$$

TSP is a rank-based method, so for each pair of genes (i, j) the "score" denoted Δ_{ij} is calculated:

$$\Delta_{ij} = |p_{ij}(C_1) - p_{ij}(C_2)| \; . \tag{4}$$

In the next step of the algorithm pairs with the highest score are chosen. There should be only one top pair in the *TSP* method, however it is possible that multiple gene pairs achieve the same top score. A secondary ranking based on the rank differences in each class and sample is used to eliminate draws.

For each top-scoring gene pair (i, j) the "average rank difference" in both C_1 and C_2 are computed and defined as:

$$\gamma_{ij}(C_1) = \frac{\sum_{n=1}^{N_1} (x_{in} - x_{jn})}{|C_1|} \; , \tag{5}$$

$$\gamma_{ij}(C_2) = \frac{\sum_{n=N_1+1}^{N} (x_{in} - x_{jn})}{|C_2|} \; . \tag{6}$$

Value of this second rank for each pair of genes (i,j) is defined as:

$$\tau_{ij} = |\gamma_{ij}(C_1) - \gamma_{ij}(C_2)| \ , \tag{7}$$

and the algorithm chooses a pair with the highest score.

The TSP classifier prediction is made by comparing the expression values from two genes (i,j) marked as "top scoring pair" with a test sample (i_{new}, j_{new}). If we observe that $p_{ij}(C_1) \geq p_{ij}(C_2)$ and $x_{inew} < x_{jnew}$, then TSP votes for class C_1, however if $x_{inew} \geq x_{jnew}$ then TSP votes for class C_2. An opposite situation is when $p_{ij}(C_1) < p_{ij}(C_2)$, cause if $x_{inew} < x_{jnew}$ TSP votes for C_1 and if $x_{inew} \geq x_{jnew}$ TSP chooses C_2. In other words, if $p_{ij}(C_1) \geq p_{ij}(C_2)$ then:

$$y_{new} = h_{TSP}(new) = \begin{cases} C_1, \text{ if } x_{inew} < x_{jnew} \\ C_2, \text{ if } x_{inew} \geq x_{jnew} \end{cases} , \tag{8}$$

where h_{TSP} is a prediction result. Opposite situation is when $p_{ij}(C_1) < p_{ij}(C_2)$.

2.2 k-Top Scoring Pairs

A *k-Top Scoring Pairs* ($k - TSP$) classifier proposed by Aik Choon Tan [20] is a simple extension of the original TSP algorithm. The main feature that differ those two methods is the number of top scoring pairs included in final prediction. In the TSP method there can be only one pair of genes and in $k - TSP$ classifier the upper bound denoted as k can be set up before the classification. The parameter k is determined by a cross-validation and in any prediction the $k - TSP$ classifier uses no more than k top scoring disjoint gene pairs that have the highest score. Both primary and secondary rankings (equations (4) and (7)) remain unchanged.

The class prediction is made by comparing the expression values for each pair of genes (i_u, j_u) $(u = 1, \ldots, k)$ with a *new* test sample. The $k - TSP$ classifier denoted as h_{k-TSP} based on partial classifiers $h_u(new)$ employs a majority voting to obtain the final prediction of y_{new}, however each vote has the same wage:

$$y_{new} = h_{k-TSP}(new) = argmax \sum_{u=1}^{k} I(h_u(new) = C_i) \ , \tag{9}$$

where $C_i \in C = \{C_1, \ldots, C_M\}$, and

$$I(h_u(new) = C_i) = \begin{cases} 1, \text{ if } \ h_u(new) = C_i \\ 0, \text{ otherwise} \end{cases} . \tag{10}$$

Meaning of $h_u(new)$ is the same as in the equation (8).

2.3 *Weight k-TSP*

In classification *Weight* $k-TSP$ proposed by us [5] all rankings have been changed, comparing to TSP and $k-TSP$. Therefore, the selection of top scoring pairs, and the prediction is different than in TSP or $k-TSP$ classifier. The main reason that motivates research on extensions of the $k-TSP$ algorithm was its limitation in finding appropriate top scoring pairs. There were two factors that could cause it. First factor that hampers finding appropriate top scoring pairs is connected to the relatively high computational complexity, which for these methods is $\theta(N*P^2)$. Microarray datasets contain huge amounts of data and the feature selection is usually applied before the actual classification. However, $k-TSP$ sensitivity to the feature selection and small size of datasets may effect rank calculations and decrease accuracy. This is connected with the second factor which is a small number of features having similar expression values and being opposite to each other in different classes.

Considering that S represents average values quotient in each pair of genes from P training samples. For each pair of genes (i,j) $(i,j \in \{1,\dots,P\}, i \neq j)$ single element from S can be described as:

$$S_{ij} = \frac{\sum_{m=1}^{N} x_{im}/x_{jm}}{N} \;. \tag{11}$$

Weight $k-TSP$ is focused on finding pairs of genes (i,j) that have the highest difference in probability of event $\{x_{in}/x_{jn} < S_{ij}\}$ $(n = 1,2,\dots,N)$ between class C_1 and C_2.

Similar to $k-TSP$, *Weigh* $k-TSP$ is a rank-based method, so for each pair of genes (i,j) score is calculated and the algorithm chooses the pairs with the highest one. Final prediction is similar to TSP or $k-TSP$ methods and involves voting. However unweighed majority voting was extended by adding weight and mixed decision rules to improve accuracy for different types of datasets.

3. *Linked TSP*

Concept of *Linked* TSP has arisen during our tests of *Weight* $k-TSP$ algorithm. We have observed that some genes, much more often to other ones, join in pairs that achieve high scores in $k-TSP$ and *Weight* $k-TSP$ rankings. This may suggest that these genes more precisely discriminate cancer samples from normal ones.

The presented method can work with any standard TSP-family classifier or different approach that calculates pairs of genes. Proposed algorithm require sorted (decreasingly by significance) list of gene pairs denoted as L from TSP methods. Idea of approach is to discover list (denoted as G) of most frequently appearing genes from list L. Let the parameter k alike in $k - TSP$ be determined by a cross-validation and stands for the upper bound on the number of top genes to be included in the final *Linked* TSP classifier.

In the first step, we seek for genes g ($g \in G = \{g_1, \dots, g_k\}$) that most frequently appears in the list L_T ($L_T \in L$). List denoted as L_T contains top T pairs of genes from L that have the highest TSP ranking. For each gene in L_T ranking is calculated based on a appearing frequency in the rest $T - 1$ pairs. After finding gene with the highest ranking score denoted as g_1, all pairs from list L_T containing this gene are removed and first step is repeated until k top genes are found.

Next step (which is optional) uses permutation of G list to remove irrelevant genes. All top genes from list G are used to test training sample - at first individually and later in double and triple size sets. At each step the worst gene or set of genes is removed. To prevent classifier over-fitting, internal 10-folds cross-validation is performed and to ensure stable results - average score of 10 runs is calculated.

Finally, algorithm determine (m-times by internal cross-validation) final number of top k genes from list G that will be used in prediction - similar to TSP method. Concept of choosing top genes is presented in Algorithm 1.

Algorithm 1 Calculate the list of $G = \{g_1, \dots, g_k\}$ top genes

```
Require: Maximum number of genes to search: $k \geq 0$
Ensure: Linked $TSP$ classifier
  for cross-validation - repeat $m$ times do
    Make an ordered list $L$ of all of the gene pairs from highest to lowest score using $TSP$ methods.
    for $i$ = 1 to $k$ do
      Make a list $L_T$ that contains top $T$ gene pairs from the $L$ list
      for each gene in $L_T$ do
        Calculate the number of pairs that involve this gene
      end for
      Add the most common gene $g_i$ to the $G$ list
      Remove every pair from $L$ that involves $g_i$
      Compute the error rate for the classifier based on genes in list $G$
    end for
  end for
  Select the value of $K$ whose average classification rate over $m$ loops is optimal.
  [optional] Remove genes from list $G$ that have the lowest accuracy through internal permutation
  Return Linked $TSP$ classifier
```

List of genes denoted as G that will be used in *Linked* TSP classifier is usually similar to the ones obtained from TSP-family classifiers. Often the top genes from each pairs that were used in *TSP* prediction also built *Linked TSP* classifier. However, no pairs of genes and the concept of "*relative expression reversals*"[8][20] in *Linked TSP* prediction model required new method to classify data.

We would like to propose rank prediction that will be composed of 3 simple steps. Let's assume that prediction model require k genes denoted as $g_1, g_2, \ldots, g_k$ from list G. Genes are marked separately in each classes $C_m \in C = \{C_1, \ldots, C_M\}$, where M denotes alike in *TSP* number of classes. Ranking for each class is presented in the equation (12).

$$Rank(C_m) = \sum_{i=1}^{k} (I_1(g_i) + I_2(g_i) + I_3(g_i)), \tag{12}$$

where:

$$I_1(g_i) = \begin{cases} \tau_1, \ where\ g_{min_i}(C_m) \leq g_i \leq g_{max_i}(C_m) \\ 0, \ otherwise \end{cases}$$

$$I_2(g_i) = \begin{cases} \tau_2, \ where\ (g_{min_i}(C_m) \leq g_{min_i}(C \setminus C_m)\ and\ g_i \leq g_{min_i}(C_m)) \\ \qquad or\ (g_{max_i}(C_m) \geq g_{max_i}(C \setminus C_m)\ and\ g_i \geq g_{max_i}(C_m)) \\ 0, \ otherwise \end{cases}$$

$$I_3(g_i) = \begin{cases} \tau_3, \ where\ |g_i - \bar{g}_i(C_m)| < |g_i - \bar{g}_i(C \setminus C_m)| \\ 0, \ otherwise \end{cases},$$

where: $g_{min_i}(C_m)$, $g_{max_i}(C_m)$, $\bar{g}_i(C_m)$ denote minimum, maximum and average value of expression level of i-gene in training dataset that was chosen for prediction, in class C_m.
Score achieved by genes from list G determines prediction result. Tested sample is classified to the class that has the highest average score through all K genes. Ranking prediction may be adjusted to analyzed dataset by parameter τ in each step.

4. Experimental Results

Performance of *Linked TSP* classifier was investigated on public available microarray datasets described in Table 1. We have comprised accuracy and size of *Linked TSP* method with *TSP*-family algorithms. In addition, other popular classifiers were analyzed and all results were enclosed in Tables from 2 to 5.

Table 1. Kent Ridge Bio-medical gene expression datasets

	Datasets	**Abbreviation**	**Attributes**	**Training Set**	**Testing Set**
1	Breast Cancer	BC	24481	34/44	12/7
2	Central Nervous System	CNS	7129	21/39	-
3	Colon Tumor	CT	6500	40/22	-
4	DLBCL Standford	DS	4026	24/23	-
5	DLBCL vs Follicular Lymphoma	DF	6817	58/19	-
6	DLBCL NIH	DN	7399	88/72	30/50
7	Leukemia ALL vs AML	LA	7129	27/11	20/14
8	Lung Cancer Brigham	LCB	12533	16/16	15/134
9	Lung Cancer University of Michigan	LCM	7129	86/10	-
10	Lung Cancer - Totonto, Ontario	LCT	2880	24/15	-
11	Prostate Cancer	PC	12600	52/50	27/8

4.1 Datasets

Datasets came from Kent Ridge Bio-medical Dataset Repository [12] and are related to studies of human cancer, including: leukemia, colon tumor, prostate cancer, lung cancer, breast cancer etc. Typical 10-folds crossvalidation was applied for datasets that were not arbitrarily divided into the training and the testing sets. To ensure stable results for all datasets average score of 10 runs is shown. All data was not transformed, no standardization or normalization was performed.

4.2 Setup

Comparison of $Linked\ TSP$ was performed with original $k-TSP$ and $Weight\ k-TSP$ algorithms. Maximum number of gene pairs k used in all prediction models was default (equal 10) through all datasets. $Linked\ TSP$ method has this number doubled because it calculates single not pairs of genes. Default values for the prediction rankings were set decreasingly: $\tau_1 = 1$, $\tau_2 = 0.5$, $\tau_3 = 0.1$ and the list L_T default size equal 100 top pairs.

All classifications were preceded by a step known as feature selection where a subset of relevant features is identified. We decided to use popular for microarray analysis method Relief-F [13] with default number of neighbors (equal 10) and 1000 features subset size.

$Linked\ TSP$ accuracy was also compared to other popular classifiers that generates comprehensible decision rules. Comparison $Linked\ TSP$ to other classifiers was performed with:

– Several popular decision trees:

   1. AD Tree (AD) - alternating decision tree
   2. BF Tree (BF) - best-first decision tree classifier
   3. J48 Tree (J48) - pruned C4.5 decision tree
   4. Random Tree (RT) - algorithm constructing a tree that considers K randomly chosen attributes at each node
   5. Simple Cart (CT) - CART algorithm that implements minimal cost-complexity pruning
- Rule classifier:
   6. JRip (JR) - rule learner - Repeated Incremental Pruning to Produce Error Reduction (RIPPER)
- Ensemble decision trees:
   7. Bagging (BG) - reducing variance meta classifier
   8. Adaboost (ADA) - boosting algorithm using Adaboost M1 method

The main software package used in the comparison experiment for these classifiers is Weka [22]. Classifiers were employed with default parameters through all datasets. Experimental results on tested datasets are described in Tables 2 and 5.

4.3 Outcome

Table 2 enclose *Linked TSP* comparison results to *TSP*-family classifiers. Since first step *Linked TSP* algorithm may involve list of best gene pairs calculated from different *TSP* methods we would like to present results separately. Let *Linked* $k-TSP$ denote results of *Linked TSP* classifier built on $k-TSP$ method and *Linked Weight* $k-TSP$ represents *Linked TSP* built on *Weight* $k-TSP$ method.

Results enclosed in Table 2 reveal that *Linked TSP* based on $k-TSP$ yield the best averaged accuracy (78.59) over 11 classification problems. *Linked TSP* based on *Weight* $k-TSP$ achieved second score and also improve *Weight* $k-TSP$ classifiers. We can observe that in 7 over 11 datasets *Linked TSP* has the highest accuracy. We believe that achieved results are promising and proposed approach may compete and be an alternative for $k-TSP$ or *Weight* $k-TSP$ methods. However in our opinion *Linked TSP* can not replace other *TSP* classifiers because each method generates significant rules that can capture interactions in datasets from different aspects. In our experiments around 50% identical genes were used by all three classifiers however the different prediction model influence the final score.

Worth to notice in Table 2 is the number of genes used in classification. It is significantly smaller in *Linked TSP* (6.57 and 11.38) to the $k-TSP$ (14.21) and *Weight* $k-TSP$ (14.85) methods. Therefore presented solution simplify decision rules and uses only significant genes in classification and prediction.

Table 2. Comparison of *Linked TSP* accuracy and size with original $k-TSP$ and $Weight\ k-TSP$ classifiers. The highest classifiers accuracy for each dataset was bolded.

Datasets	**Classifiers**							
	k-TSP		Weight k-TSP		Linked k-TSP		Linked Weight k-TSP	
	accuracy	size	accuracy	size	accuracy	size	accuracy	size
1. BC	74.73	17.20	47.36	18.00	**88.42**	8.10	51.57	12.20
2. CNS	59.49	17.96	51.16	17.84	55.66	7.96	**65.33**	18.37
3. CT	76.83	10.40	85.47	5.08	82.19	8.37	**86.73**	18.13
4. DS	78.90	14.91	62.20	17.68	65.65	8.37	**86.95**	15.52
5. DF	**91.44**	17.44	81.41	15,71	87.80	8.44	87,76	13.11
6. DN	56.37	17.60	52.25	17.60	**70.00**	9.00	51.62	20.00
7. LA	**94.11**	18.00	93.23	17.60	91.17	3.00	91.17	2.00
8. LCB	77.18	2.00	96.71	16.80	**100.00**	3.00	91.94	2.00
9. LCM	95.62	15.48	98.53	15.34	97.26	3.71	**99.26**	2.00
10. LCT	73.41	12.44	**89.50**	4.76	67.00	8.28	74.50	13.27
11. PC	63.66	12.88	**73.66**	16.92	59.33	4.08	59.33	8.57
Average score	76.52	14.21	75.59	14.85	**78.59**	6.57	76.92	11.38

Table 3. Marker genes used in Lung Cancer Brigham dataset classification

Classifiers	k-TSP	Weight k-TSP	Linked k-TSP	Linked Weight k-TSP
Accuracy	77.18%	96.71%	100.00%	91.94%
Genes	31393_r_at **33499_s_at**	more than 20 genes	37947_at **33499_s_at_at** 36528	37013_at 35236_g_at

In one of our experiments we tested Lung Cancer dataset (*LCB*) from "Brigham and Women's Hospital" Harvard Medical School. We managed to achieve perfect accuracy with only 3 genes to other classifiers results (90-99%) described in [9] that used 6 features and more. In Table 3 we have enclosed genes that build tested classifiers and we have bolded identical ones. We may observe that using more features in this case increased classifiers accuracy. However involving too many genes in decision model makes the method harder to understand by human experts.

Higher number of genes used in prediction not always cause increase of an accuracy. Different Lung Cancer dataset (*LCM*) from University of Michigan [1] may be a good example. Number of genes that build classification model in *Linked TSP* was more than 5 times smaller to *TSP* methods although accuracy results slightly increased. In Table 4 we have compared genes used by $Linked\ k-TSP$ and $Linked\ Weight\ k-TSP$ classifiers. Higher number of genes to classifiers size is caused by crossvalidation of training dataset as no tested set was provided. Similar genes that

Table 4. Marker genes used in Lung Cancer University of Michigan dataset classification

Classifiers	k-TSP	Weight k-TSP	Linked k-TSP	Linked Weight k-TSP
Accuracy	95.62%	98.53%	97.26%	99.26%
Genes	more than 40 genes	more than 40 genes	**J03600_at** **M24486_s_at** **X64177_f_at** **U87964_at** U60061_at Y09216_at	**J02871_s_at** **M24486_s_at** **X64177_f_at** **U87964_at** - -

build *Linked TSP* classifiers despite different algorithms suggest that those genes can be considered as marked. Original *TSP* methods used over 40 different genes in prediction. They contained *Linked TSP* genes however many irrelevant features were also enclosed making the classifiers results much more difficult to analyze and interpret by human experts.

We have observed that genes from list *G* that built *Linked TSP* more often occurred in tested decision trees and the rest of classifiers to selected ones from $k-TSP$ or *Weight* $k-TSP$. This may confirm that *Linked TSP* prediction model involve only predominatingly genes from *TSP* pairs. Relying on experimental results we may conclude that *Linked TSP* simplify decision rules without reducing classification accuracy and even improving it for some datasets.

In our research we also investigate performance 8 different classifiers on datasets from Table 1. In our research we were focused on the "white box" methods rather the "black box" algorithms and this is the reason why methods like Support Vector Machine (SVM) [**?**] or neutral networks [4] were not included in our analysis. Comparison tests were performed with methods that like TSP generate simple and comprehensible decision rules. Results for those classifiers are enclosed in Table 5. If we compare them with ones from Table 2 we may observe that *TSP*-family classifiers achieve relatively higher average accuracy through all datasets. Even methods, that generate more complex to analyze and interpret decision tree ensembles like Bagging or Boosting also achieved slightly lower score.

5. Conclusion and Future Works

This paper presents extension of *TSP*-family classifiers called *Linked TSP*. We believe it is an interesting approach that may compete with *TSP*-family methods. General improvement of *Linked TSP* method did not exceed 2% although for some analyzed datasets idea of linking top pairwise mRNA comparisons of gene expressions

Table 5. Comparison classifiers accuracy

Dataset/Classifier	1. AD	2. BF	3. J48	4. RT	5. CT	6. JR	7. BG	8. ADA
1. BC	42.10	47.36	52.63	36.84	68.42	73.68	63.15	57.89
2. CNS	63.33	71.66	56.66	63.33	73.33	65.00	71.66	75.00
3. CT	74.19	75.80	85.48	70.96	75.80	74.19	79.03	79.03
4. DS	95.74	80.85	87.23	68.08	82.97	74.46	87.23	89.36
5. DF	88.31	79.22	79.22	81.81	83.11	77.92	85.71	90.90
6. DN	50.00	60.00	57.50	53.75	62.50	61.25	58.75	65.00
7. LA	91.17	91.17	91.17	55.88	91.17	94.11	94.11	91.17
8. LCB	81.87	89.65	81.87	77.18	81.87	95.97	82.55	81.87
9. LCM	96.87	96.87	98.95	91.66	96.87	93.75	97.91	96.87
10. LCT	69.23	61.53	58.97	53.84	58.97	64.10	61.53	69.23
11. PC	38.23	44.11	29.41	47.05	44.11	32.35	41.17	41.17
Average score	71.90	73.04	72.05	65.02	75.65	74.30	76.68	**76.72**

increased accuracy for over 10%. The size of classification model was significantly smaller (almost 40%) therefore *Linked TSP* generates more adequate and comprehensible decision rules. However, for some tested datasets original *TSP* was more accurate that is why the best *TSP* method can not be indicated. It is worth to notice that all *TSP* classifiers used similar set of top genes in decision model. This may suggest that each algorithm generates significant rules that capture interactions in datasets from different aspects.

Classification results comparison through tested datasets reveal that *TSP*-family classifiers are good alternative to decision trees and other classification rules. We believe that there is still place for improvement *TSP*-family classifiers. Merging the $k-TSP$, *Weight* $k-TSP$ and *Linked TSP* predictive power in a single algorithm might significantly increase accuracy and provide efficient decision rules with clear biological connections to adequate cancer type.

References

[1] Beer D.G.: Gene-expression Profiles Predict Survival of Patients with Lung Adenocarcinoma. Nature Medicine, 8(8):816-823, 2002.

[2] Bittner M., Meltzer P., Chen Y.: Molecular classification of cutaneous malignant melanoma by gene expression profiling. Nature, 406, 536-540, 2000.

[3] Brown P.O., Botstein D.: Exploring the new world of the genome with DNA microarrays. Nature Genet 21. 33-37, 1999.

[4] Cho H.S., Kim T.S., Wee J.W.: cDNA Microarray Data Based Classification of Cancers Using Neural Networks and Genetic Algorithms. Nanotech, vol. 1, 2003.

[5] Czajkowski M., Krętowski M.: Novel extension of k-TSP algorithm for microarray classification. Lecture Notes in Artificial Intelligence, vol. 5027:456-465, 2008.

[6] Dhanasekaran S.M.: Delineation of prognostic biomarkers in prostate cancer. Nature, 412, 822-826, 2001.

[7] Duggan D.J., Bittner M., Chen Y., Meltzer P., Trent J.M.: Expression profiling using cDNA microarrays. Nature Genetics Supplement, 21, 10-14, 1999.

[8] Geman, D., d'Avignon, C., Naiman, D.Q., Winslow, R.L.: Classifying Gene Expression Profiles from Pairwise mRNA Comparisons. Statistical Applications in Genetics and Molecular Biology, 3(1), 19, 2007.

[9] Gordon J.G.: Translation of Microarray Data into Clinically Relevant Cancer Diagnostic Tests Using Gege Expression Ratios in Lung Cancer And Mesothelioma. Cancer Research, 62:4963-4967, 2002.

[10] Grześ M., Krętowski M.: Decision Tree Approach to Microarray Data Analysis. Biocybernetics and Biomedical Engineering, vol. 27(3), 29-42, 2007.

[11] Hastie T., Tibshirani R., Friedman J.H.: The Elements of Statistical Learning. Springer-Verlag, 2001.

[12] Kent Ridge Bio-medical Dataset Repository: http://datam.i2r.a-star.edu.sg/datasets/index.html

[13] Kononenko I.: Estimating Attributes: Analysis and Extensions of RELIEF. In: European Conference on Machine Learning, 171-182, 1994.

[14] Lockhart D.J., Winzeler E.A.: Genomics, gene expression and DNA arrays. Nature 405, 827-836, 2000.

[15] Lu Y., Han J.: Cancer classification using gene expression data. Information Systems, 28(4), pp. 243-268, 2003.

[16] Nelson P.S.: Predicting prostate cancer behavior using transcript profiles. Journal of Urology, 172, 28-32, 2004.

[17] Sebastiani P., Gussoni E., Kohane I.S., Ramoni M.F.: Statistical challenges in functional genomics. Statistical Science, 18(1), 33-70, 2003.

[18] Simon R., Radmacher M.D., Dobbin K., McShane L.M.: Pitfalls in the use of DNA microarray data for diagnostic and prognostic classification. Journal of the National Cancer Institute, 95, 14-18, 2003.

[19] Speed T.: Statistical Analysis of Gene Expression Microarray Data. Chapman & Hall/CRC, New York, 2003.

[20] Tan A.C., Naiman D.Q., Xu L., Winslow R.L. and Geman D.: Simple decision rules for classifying human cancers from gene expression profiles. Bioinformatics, vol. 21, 3896-3904, 2005.
[21] Vapnik, V.N.: Statistical Learning Theory. Wiley, New York, 1998.
[22] Witten I.H., Frank E.: Data Mining: Practical machine learning tools and techniques. 2nd edn. Morgan Kaufmann, San Francisco, 2005.

ROZSZERZENIE METOD Z RODZINY TSP W KLASYFIKACJI MIKROMACIERZY DNA

Streszczenie Klasyfikacja danych mikromacierzowych a także późniejsza interpretacja reguł decyzyjnych może być skutecznie przeprowadzona za pomocą metod z rodziny Top Scoring Pair, polegających na analizie par genów o przeciwstawych poziomach ekspresji w różnych klasach. W poniższym artykule zaprezentowano nową metodę: *Linked TSP*, która rozszerza działanie klasyfikatorów $k - TSP$ i *Weight* $k - TSP$. W przeciwieństwie do algorytmów z rodziny TSP proponowane rozwiązanie tworzy reguły decyzyjne zbudowane z pojedynczych genów co znacznie ułatwia ich późniejszą interpretacje medyczną. W algorytmie wykorzystywane są pary genów uzyskane z algorytmów TSP z których następnie, wybierane są pojedyncze, najczęściej powtarzające się geny. Testy algorytmu *Linked TSP* przeprowadzone zostały na rzeczywistych zbiorach danych pacjentów a uzyskane wyniki są obiecujące.

Słowa kluczowe: klasyfikacja par genów zależnych, analiza mikromacierzy, reguły decyzyjne, ekspresja genów

Artykuł zrealizowano w ramach pracy badawczej W/WI/5/08.

ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2009
Informatyka – Zeszyt 4

Joanna Gościk[1], Józef Gościk[2]

NUMERICAL EFFICIENCY OF THE CONJUGATE GRADIENT ALGORITHM - SEQUENTIAL IMPLEMENTATION

Abstract: In the paper we report on a second stage of our efforts towards a library design for the solution of very large set of linear equations arising from the finite difference approximation of elliptic partial differential equations (PDE). Particularly a family of Krylov subspace iterative based methods (in the paper exemplified by the archetypical Krylov space method - Conjugate Gradient method) are considered. The first part of the paper describes in details implementation of iterative algorithms for solution of the Poisson equation which formulation has been extended to the three-dimensional. The second part of the paper is focused on the performance measurement of the most time-consuming computational kernels of iterative techniques executing basic linear algebra operations with sparse matrices. The validation of prepared codes as well as their computational efficiency have been examined by solution a set of test problems on two different computers.

Keywords: Iterative solvers, Finite difference method, Poisson equation, Performance of sequential code

1. Introduction

At this time, the search for high performance in a large scale computation is of growing importance in scientific computing. This is very important due to recent attention which has been focused on distributed memory architectures with particular interest in using small clusters of powerful workstations. However, the appearing of modern multiprocessors platforms demands to resolve a lot of tasks which are addressed to efficient porting solvers on three different architectures: a sequential machine, a shared memory machine and on a distributed memory machine. So the key problem is to understand how the computation aspects of using linear algebra solvers (in our case - iterative) can affect the optimal execution of code on machines with different architecture.

For understanding the implication of design choices for systems, and for studying increasingly complex hardware architecture and software systems very important

[1] Faculty of Computer Science, Bialystok Technical University, Bialystok, Poland
[2] Faculty of Mechanical Engineering, Bialystok Technical University, Bialystok, Poland

tools are more or less standardized benchmarks. However running for example very well known Linpack benchmark [1] on complex architectures is not so obvious and easy in interpretation. The problem was discovered when the execution time of Linpack was studied systematically on different machines (the results and main problems arise when we tried to estimate performance of the benchmark will be pulished elsewhere). This observation in our opinion should be interpreted by misunderstanding the impacts of different architecture designs. Among them are characteristic differences in latency between different levels of caches and memory, introduction of simultaneous multithreading and single-chip multiprocessors, to mention of few.

In the paper we try to explore some of the main challenges in implementation and estimation computational efficiency of the executing sequential code. The special attention has been paid on credible study of performance in terms of number of equations. The paper report on continuation of our research described in [2]. So, we were mainly interested in building and improvement of the software for solving very large set of linear equations arising from the finite difference approximation of the elliptic PDE. At the stage the iterative solver module has been rearranged in a such way which allows flexible operating on separate basic algebra operations as inner products of two vectors, updating of vectors and the matrix-vector multiplication.

This paper continues as follows. In Section 2 we present the scope of the mathematical formulation of the elliptic PDE as well as results of calculations made by making use raw, not tuned codes. Our main goal was to prepare validated iterative solver for systems with large sparse matrices of coefficients. In Section 3 we describe in details our data structures and results of testing performance on two platforms (including one which is installed in WI Bialystok - Mordor2). In Section 4 we present our results for timing and estimating performance of running different parts of the solver routines. Finally, in Section 5 we summarize our efforts and formulate direction of further research.

2. Software for the Poisson equation solution

In mathematical formulation we confined ourselves to the one of the simplest elliptic boundary value problem described by the Poisson equation in a classical form (for the constant coefficients)

$$\nabla^2 u(x) = f(x) \quad \forall (x) \in \Omega \tag{1}$$

where u is a continuous function, ∇^2 is the Laplace operator and f is specified in domain Ω forcing (source). The problem (1) is considered with homogenous Dirichlet boundary conditions

$$u = 0 \quad \forall (x) \in \partial\Omega \tag{2}$$

The essential difference of the work in the relation to the previous stage [2] is extension of the consideration on any descryption in space. Taking respectively 1D, 2D and 3D domains we worked out three different codes (Pois1D, Pois2D and Pois3D) dedicated for solution of these problems. For the purpose of the codes validation, three different problems have been resolved. However the special attention is also directed on linear algebra problems arising from approximation technique. Each continuous problem has been discretized by standard second order difference method on structured grid. Thus our numerical schemes generate a class - in general very large systems - of linear equations which can be expressed as

$$A_{\_D(h)} \cdot \hat{u} = f_h \tag{3}$$

where $A_{\_D(h)}$ is the matrix of coefficients which structure depends on finite difference operator used, û is the vector of nodal unknowns, and f_h is the right - hand side vector.

The system of equations (3) is solved using the conjugate gradient solver with or without diagonal scaling (preconditioning). The iteration were performed till the relative norm of residual [2] dropped to 10^{-8}.

The accuracy of the schemes were analyzed by convergence of a global measure of the error, which a discrete norm (L2 - norm) has been chosen as

$$||e|| = \left(\sum_J e_J^2 / N_{max} \right)^{1/2}, \tag{4}$$

where N_{max} is the total number of grid points and J stands for summation index as a i, ij, ijk for 1D, 2D and 3D respectively.
The error (4) is defined as

$$e = u - \hat{u} \tag{5}$$

and overall accuracy is found in each case by a systematic grid refinement study.

In the next, we present our numerical experiments which were carried out for three representative test problems. Our implementation is in C++.

2.1 Case 1D

As a first test case we consider (1,2) in 1D domain. In this case the problem in fact reduces to ordinary differential equation, which consequently for general Laplace operator form we defined as a

$$-\frac{d^2u}{dx^2} = f(x) \quad \forall (x) \in \Omega \tag{6}$$

where Ω is defined as the open section $(0,1)$. For homogenous Dirichlet boundary conditions $u(x=0)=0$ and $u(x=1)=0$ the problem has the exact solution

$$u(x) = (1-x)\cdot x\cdot e^{x}$$

with the source term defined as

$$f(x) = (3+x)\cdot x\cdot e^{x}. \tag{7}$$

For finite difference based solution of the Case 1D we use uniform grid describing the computational domain Ω. The grid nodes are equally spaced, $h=1/(n_x-1)$, where n_x is the number of grid points in x-direction. The grid coordinates are defined as

$$x_i = (i-1)\cdot h, \quad \text{for} \quad 1 \le i \le n_x .$$

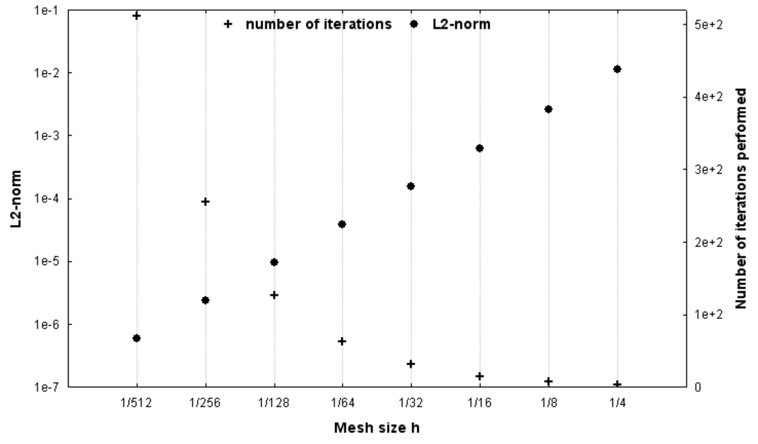


Fig. 1. Convergence plot (in L2-norm) and number of iterations performed versus mesh size for the test Case 1D.

We solved the problem (6, 7) on a series of meshes with h = 1/4, 1/8, 1/16, 1/32, 1/64, 1/128, 1/256 and 1/512. The results in a form of norm (4) for the sequence of meshes are presented in Fig. 1.

2.2 Case 2D

As a second test case we consider the Poisson equation defined for 2D. Laplace operator modifies to the form

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = f(x,y) \quad \forall (x,y) \in \Omega, \tag{8}$$

where Ω is defined as a unit square $(0,1)\times(0,1)$, and the exact solution has a form

$$u(x,y) = \sin(\pi x)\cdot\sin(2\pi y)\ .$$

For taken form of the exact solution and boundary conditions kind of (2) the source term is defined as

$$f(x,y) = 5\pi^2\cdot\sin(\pi x)\cdot\sin(2\pi y)\ . \qquad (9)$$

For finite difference based solution of the Case 2D we use uniform grid describing

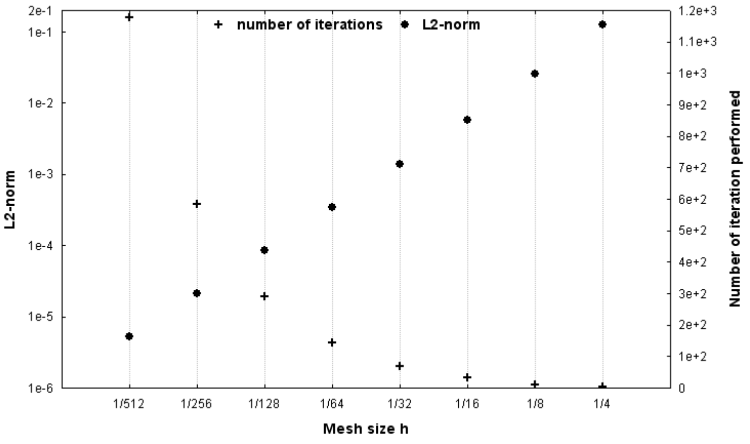


Fig. 2. Convergence plot (in L2-norm) and number of iterations performed versus mesh size for the test Case 2D.

the computational domain Ω. The grid nodes are equally spaced, $h = 1/(n_x - 1) = 1/(n_y - 1)$, where n_x and n_y is the number of grid points in x-, and y-direction, respectively. For unifirm mesh, the grid coordinates are defined as

$$x_i = (i-1)\cdot h,\quad y_j = (j-1)\cdot h \quad \text{for} \quad 1 \le i,j \le n(= n_x = n_y)\ .$$

The results for the same sequence of meshes in the 2D case as for the case 1D (uniform spacing along x- and y-direction) are presented in Fig. 2.

2.3 Case 3D

Finally, we successfully extended finite difference schemes approximation to 3D domains, so the generic Poisson equation has a form

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} - \frac{\partial^2 u}{\partial z^2} = f(x,y,z) \quad \forall (x,y,z) \in \Omega \qquad (10)$$

where Ω is defined as the unit cube $(0,1)\times(0,1)\times(0,1)$. For testing purpose we again took a test problem which has an exact solution

$$u(x,y,z) = \sin(\pi x)\cdot\sin(\pi y)\cdot\sin(\pi z)\ .$$

For the homogenous Dirichlet boundary conditions (2) - corresponding source term is defined as

$$f(x,y,z) = 3\pi^2 \sin(\pi x)\cdot\sin(\pi y)\cdot\sin(\pi z). \tag{11}$$

Also in this case, for defining the computational domain, we use an uniform, rectangular finite difference grid. The grid nodes are again equally spaced, $h = 1/(n_x - 1) = 1/(n_y - 1) = 1/(n_z - 1)$, where in general n_x, n_y, n_z are the number of grid points in x-, y-, and z-direction, respectively. For uniform spacing ($n_x = n_y = n_z = n$) the grid coordinates are defined as

$$x_i = (i-1)\cdot h,\quad y_j = (j-1)\cdot h,\quad z_k = (k-1)\cdot h,\quad 1 \le i,j,k \le n\ .$$

The problem Case 3D has been resolved by using different discrete problem with $h = 1/4, 1/8, 1/16, 1/32, 1/64, 1/128$. Results are given in Fig. 3.

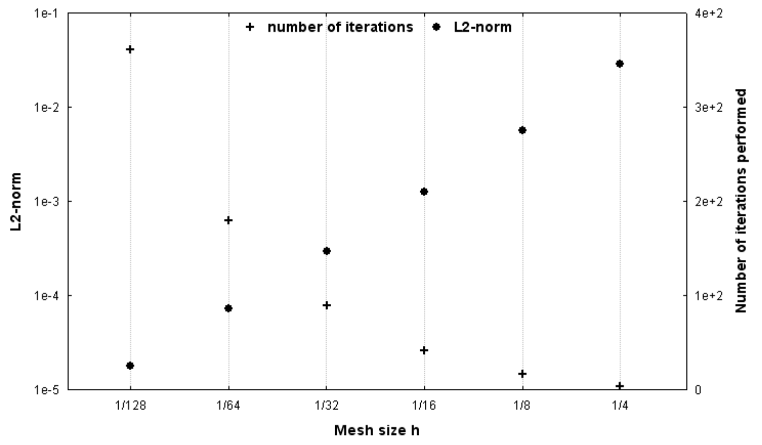


Fig. 3. Convergence plot (in L2-norm) and number of iterations performed versus mesh size for the test Case 3D.

A log-log plots (Fig. 1, Fig. 2, Fig. 3) of the error in L2 - norm versus h is approximately a straight line with a constant slope when h is sufficiently small. This observation confirm correctness of the worked out numerical schemes. At the same Figures we also present the total number of iterations performed for each run. Points related to performed iterations versus mesh size illustrate also the typical behavior of CG iterative algorithm for which number of iterations grows as the problem size increases. Moreover, they are also clear confirmation that CG like algorithm has a finite termination property - that means that at N-th iteration the algorithm will terminate (for our boundary value problem N = n-2). For Case 1D this property is perfectly fulfilled. For 2D and 3D cases the number of iterations needed for convergence is likely much lower than N.

3. Performance and preliminary timing experiments

The obtained results very well validate the codes Pois1D, Pois2D and Pois3D. In this way - especially by comparisons with analytical solutions - we ensure that we operate on validated software. On this base we can redirect the main attention to linear algebra problems arising from approximation technique. It is worth to notice, that in the previous Section we have illustrated convergence plots as well iteration counts only in the sense of the maximum of iteration needed to obtain convergence.

Before we describe in details iterative solvers performance lets take a look on the linear algebra demands. As was pointed earlier, if we will consider a standard second order finite difference discretization of (1) on regular grid in 1D, 2D, and 3D dimensions we obtain linear system of equations (3). Data structure for storing grid points is not important because uniform mesh subdivision has been assumed. Quite different situation is when we must decide about data structure for storing coefficients of the matrix A. Strictly, after discretization on uniform mesh, the A is a matrix which is symmetric, positive definite and has a block-tridiagonal structure where the block have order N. In details, depending on the problem under consideration, the coefficient matrix is described in Tab. 1

Table 1. Coefficient matrix forms in diagonal format.

A_{1D}	has a 3-diagonal structure, in which non-zero coefficients (2) lie on the main diagonal and coefficients equal (-1) lie on the two off-diagonals;
A_{2D}	has a 5-diagonal structure, in which all the non-zero coefficients lie on: the main diagonal (4), the two neighboring diagonals (-1), and two other diagonals (-1) removed by N positions from the main diagonal;
A_{3D}	has a 7-diagonal structure, in which all the non-zero coefficients lie on: the main diagonal (6), the two neighboring diagonals (-1), two diagonals (-1) removed by N positions from the main diagonal and two other diagonals (-1) removed by N×N positions from the main diagonal.

Our final choice of data structure for matrices was primarily dictated by goals taken among of them first of all we wanted to operate on a very large sets of data. In result, in spite of the true sparsity of matrices A we store them in a two different formats

1. dense - which is standard practice for storing full matrices, and
2. diagonal [3] - which is a most preferred for the diagonal matrices resulting from finite difference discretization technique.

In this way we can easily operate on the diagonal scheme which may allow the computational problem to be kept in main memory, and dense array scheme which can not to be kept in main memory. For this, in fact we created two versions of our codes for each Poisson problem, namely: Pois1D(_DNS, _DIA), Pois2D(_DNS, _DIA) and Pois3D(_DNS, _DIA). The needed resources for storage of the matrix coefficients have been summarized in Tab.2.

Table 2. Size of coefficient matrices for different formats of storage.

	Dense format		**Diagonal format**			
		Matrix size	Diagonals		Matrix size	Problem size
1D	$N \times N$	261121	3	$N \times 3$	1533	511
2D	$N^2 \times N^2$	68184176641	5	$N^2 \times 5$	1305605	262144
3D	$N^3 \times N^3$	887503681	7	$N^3 \times 7$	14338681	2048383

Next, in this section we present preliminary results of possible performance of basic algebra operations attainable on the tested machines. The summary of technical specification of the used machines is given in Tab. 3.

Table 3. Machines used in tests.

Name	**PC**	**Mordor2 cluster**
CPU	Intel Core 2 Duo E6600	Intel Xeon X5355
OS	Ubuntu 7.10	CentOS 4.6
C compiler	gcc ver. 4.1.3	gcc ver. 3.4.6
C flags	no optimization	no optimization
BLAS/*	in line/**	in line/**

/*	Basic Linear Algebra Subprograms (BLAS) - proposed by Lawson et al. [4], at present optimized and implemented in a form of kernel routines of different specialized linear algebra packages.
/**	The results presented in the paper were obtained by running the simple loops as an in-line code.

The point of the first experiments was to get basic information about instruments for timing codes running on Linux platforms and especially to find attainable timer resolution. Because there are several Linux timers as: *clock*, *getrusage*, *clock_gettime*, *gettimeofday* in the first experiment we performed a several tests in order to chose the one with the best resolution and which invoking give the most stable results. For this we looked for the elapsed time in calculation one of the simplest algebraic operation (vector updating isolated to the one element)

$$x = x + \alpha \cdot y, \tag{12}$$

where x, y and α are known numbers (in double precision).

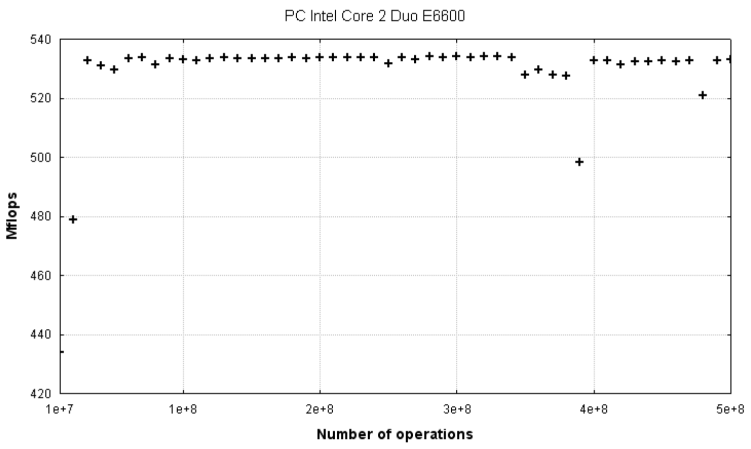


Fig. 4. Performance chart for simple updating - PC Intel Core 2 Duo E6600.

Because the timers have a different resolution (declared µs, 1 ms, 1/100th, 1s) therefore the timing has been obtained by execute calculation many times (repetitions). As a result of this preliminary set of experiments we decided to use for timing the C *gettimeofday* function which provides accuracy to approximately 1µs. Using the function we estimate performance of the used machines by repetitions of (12) from 10^7 to the $5 \cdot 10^7$ times. The result of these calculations are given in Fig. 4 and Fig. 5.

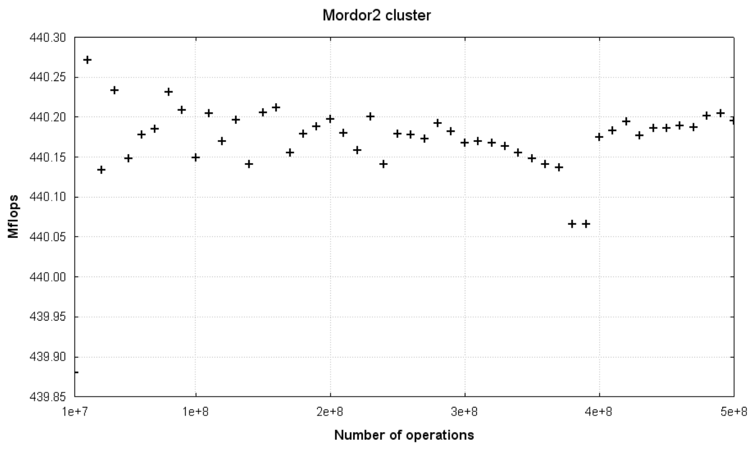


Fig. 5. Performance chart for simple updating - Mordor2 cluster Intel Xeon X5355.

4. Iterative solver performance

Among of many solution techniques for symmetric systems the conjugate gradient (CG) method is an ideal iterative solver. The CG algorithm can be also treated as a generic for all family of Krylov subspace methods. The CG algorithm in your classical form which we will called next as CGHS, is well known and his pseudo-code is given by Algorithm 1 [3].
As we can see in a sequence of CGHS algorithm there are three basic linear algebra operations:

1. `_dot` - scalar (inner or dot) product. There are two `_dot`'s in the algorithm what gives 4·n operations.

$$s = x \cdot y = (x, y) = \sum_{i=1}^{n} x_i \cdot y_i,$$

2. `_axpy` - vector update (where underscore mark in the operation name is used according to known from BLAS terminology prefix convention). There are three `_axpy`'s in the algorithm what gives 6·n operations.

$$y = y + \alpha \cdot x\,,$$

where x and y are vectors of length n, and α is a number.

3. `matvec` - matrix and vector multiplication. There are one `matvec` in the algorithm what gives $2 \cdot n^2$ operations.

$$y = Ax\ ,$$

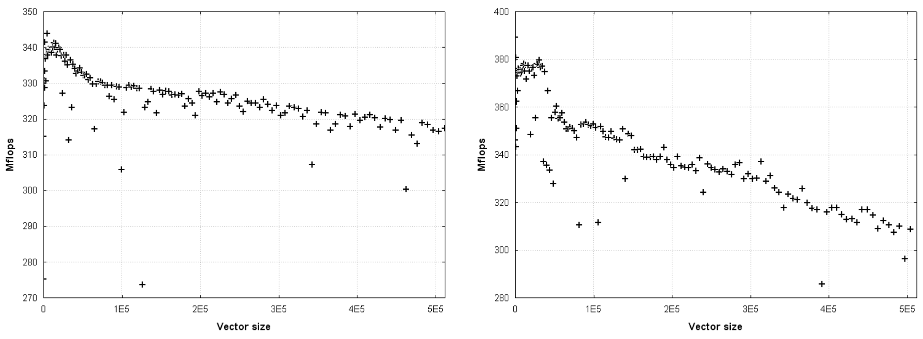
where A in our case is two-dimensional array which is $A_{n\times n}$ in dense format and $A_{n\times d}$ - banded, (incorporated diagonal format storage).

Table 4. Number of memory references and floating point operations for vectors of length n [5].

	Read	Write	Flops	Flops/mem access
_dot	$2 \cdot n$	1	$2 \cdot n$	1
_axpy	$2 \cdot n$	n	$2 \cdot n$	$\frac{2}{3}$
matvec	$n^2 + n$	n	$2 \cdot n^2$	2

Table 5. System parameters.

System parameters	PC Intel Core 2 Duo E6600	Mordor2 cluster
Clock rate	2.4 GHz	2.66 GHz
Bus speed	1066 MHz	1333 MHz
L1 cache	64 KB	128 KB
L2 cache	4 MB	8 MB

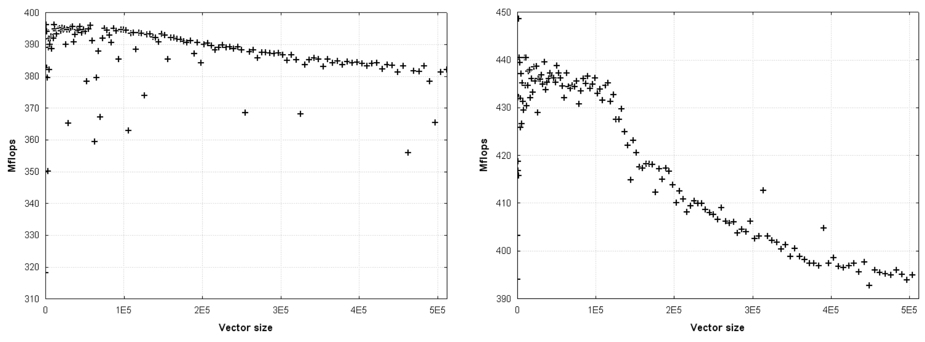


(a) PC Intel Core 2 Duo E6600 (b) Mordor2 cluster Intel Xeon X5355

Fig. 6. Daxpy

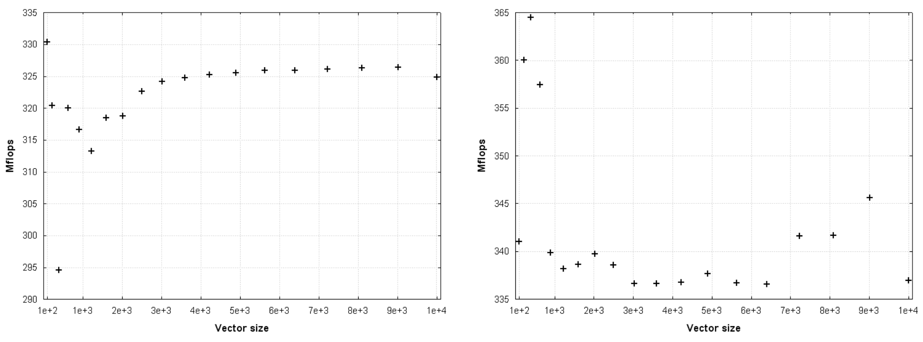
We timed the solver in two stages. First stage refers to the execution time which accounts only for the solver basic algebraic operation (`_axpy`, `_dot` and `matvec` for different format of matrices storage). The results are given in Figures 6÷9. All results in the Figures clearly show that exist some kind of mismatch between CPU and memory performance. This mismatch has an unfavorable influence on computer performance because as a general CPU's outperforms memory systems. The system parameters of the platform/machines we use in the paper are characterized in Tab. 5. The Mflops rates reflect also the ratios of memory access of the two machines. The high rates are for vectors that can be held in the on-chip cache. The low rates with very long vectors are related to the memory bandwith. The matvec has about the same performance as daxpy if the matrix can be held in cache. Otherwise, it is considerably reduced.

Next, in the second stage we measured the overall performance of the CGHS routine according to the scheme given in Algorithm 1. The timing covers entire iteration process which takes into account the time spent by the program/code for executing all commands and instructions.



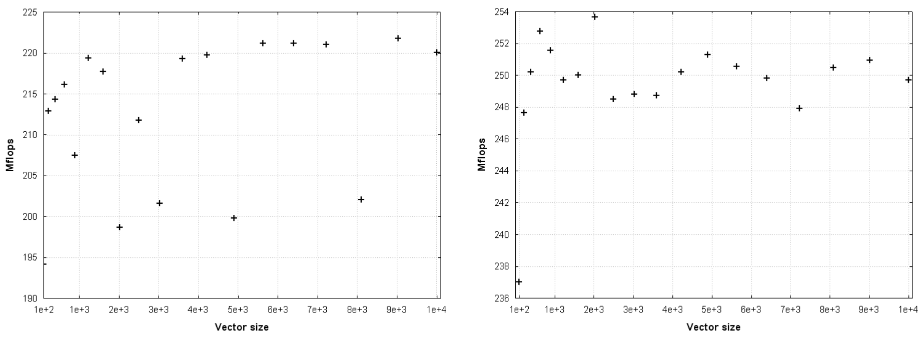
(a) PC Intel Core 2 Duo E6600 (b) Mordor2 cluster Intel Xeon X5355

Fig. 7. Ddot



(a) PC Intel Core 2 Duo E6600 (b) Mordor2 cluster Intel Xeon X5355

Fig. 8. Matvec DNS



(a) PC Intel Core 2 Duo E6600 (b) Mordor2 cluster Intel Xeon X5355

Fig. 9. Matvec DIA

5. Summary

In the paper, we described the implementation of the iterative, conjugate gradient based solver for the Poisson equation. The special attention has been paid on estimate of performance of the most time consuming parts of the code.

Summarizing our achievements we compare our performance results to theoretical peak performance of the two, various platforms used in the experiments. As a peak performance we take counted number of floating point additions and multiplications which can be completed in a period of machine cycle time. So, during the one cycle we can estimate theoretical peak performance as a

$$\frac{1\text{operation}}{1\text{cycle}} \cdot \text{cycletime} = \text{peak performance}\,[\text{Mflop/s}].$$

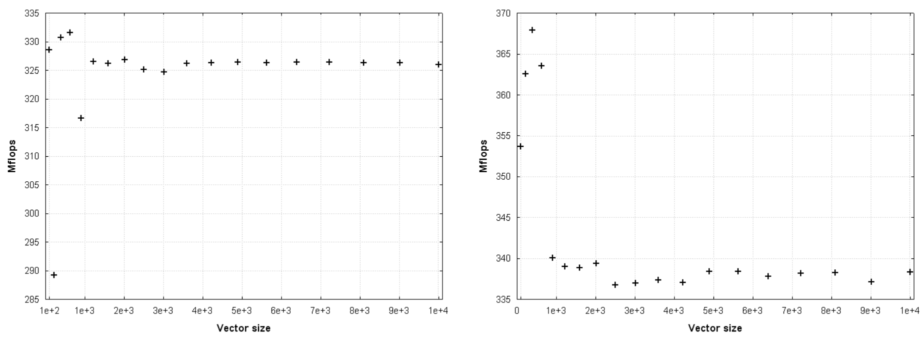
Table 6 shows comparison of the used computers peak performance and attained, average performance of the CGHS solver. It is clear, that at the stage we can say

Table 6. Summary of the CGHS efficiency results.

Machine	**Peak performance** [Mflop/s]	**CGHS performance** [Mflop/s]	**CGHS efficiency** [%]
PC Intel Core 2 Duo E6600	2400	285 ÷ 335	12 ÷ 14
Mordor2 cluster	2600	335 ÷ 370	13 ÷ 14

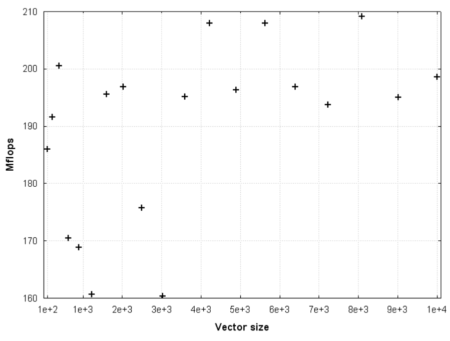
only that we posses unoptimized, correct sequential codes for solution the Poisson equation (Pois1D, Pois2D, Pois3D). At the further stages of the project we would like to do the following extensions of the current implementation, by

- adopting or preparing own instrumentation necessary to the baseline performance measurement (including possible implementation of a free available profilers),
- performing compiler optimization with the special emphasis on free available gcc compilers,
- linking optimized libraries of the BLAS kernels and asses the possible improvements in attainable performance,
- modification of source solver code taking into account possible identification of the performance bottlenecks.

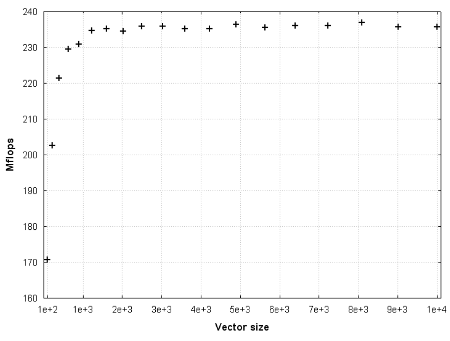


(a) PC Intel Core 2 Duo E6600 (b) Mordor2 cluster Intel Xeon X5355

Fig. 10. Overall performance of the CGHS routine - dense matrix of coefficients



(a) PC Intel Core 2 Duo E6600



(b) Mordor2 cluster Intel Xeon X5355

Fig. 11. Overall performance of the CGHS routine - diagonal matrix of coefficients

Algorithm 1 The standard (unpreconditioned) CGHS algorithm

```
$k = 0$
$x_0 = 0$
**if** $x_0 \neq 0$ **then**
    $r_0 = Ax_0 - b$
**else**
    $r_0 = b$
**end if**
$\rho_0 = ||r_0||^2$
**while** $\sqrt{\rho_k} > \varepsilon \cdot ||r_0||$ **do**
    **if** $k = 0$ **then**
        $p_1 = r_0$
    **else**
        $p_{k+1} = r_k + (\rho_{k-1}/\rho_k) \cdot p_k$   // _axpy
    **end if**
    $k = k + 1$
    $w_k = A \cdot p_k$   // matvec
    $\alpha_k = \rho_{k-1}/p_k^T \cdot w_k$   // _dot
    $x_k = x_{k-1} + \alpha_k \cdot p_k$   // _axpy
    $r_k = r_{k-1} - \alpha_k \cdot w_k$   // _axpy
    $\rho_k = ||r_k||^2$   // _dot
**end while**
$x = x_k$
```

References

[1] Dongarra J., Luszczek P., Petitet A.: The LINPACK Benchmark: Past, Present, and Future. Concurrency and Computation: Practice and Experience, Vol. 15, No. 9, 2003, pp. 803-820.
[2] Gościk J., Gościk J.: Numerical efficiency of iterative solvers for the Poisson equation using computer cluster. Zeszyty Naukowe Politechniki Białostockiej, Seria: Informatyka, Vol. 3, 2008.
[3] Saad Y.: Iterative Methods for Sparse Linear Systems. Second Edition, SIAM, Philadelphia, Pa, 2003.
[4] Lawson C.L, Hanson R.J., Kincaid D.R., Krogh F.T.: Basic linear algebra subprograms for Fortran usage. ACM Transactions on Mathematical Software, Vol. 5, No. 3, September 1979, pp. 308-323.
[5] Arbenz P., Peterson W.: Introduction to Parallel Computing - A Practical Guide with examples in C. Oxford University Press, 2004, Series: Oxford Texts in Applied and Engineering Mathematics No. 9, Oxford 2004.

EFEKTYWNOŚĆ NUMERYCZNA ALGORYTMU GRADIENTÓW SPRZĘŻONYCH - IMPLEMENTACJA SEKWENCJNA

Streszczenie: Przedstawiono wyniki realizacji drugiego etapu projektu mającego na celu opracowanie i wdrożenie algorytmów rozwiązywania wielkich układów równań liniowych generowanych w procesie aproksymacji eliptycznych równań różniczkowych o pochodnych cząstkowych (PDE) metodą różnic skończonych. W szczególności skoncentrowano się na implementacji wersji sekwencyjnej najbardziej reprezentatywnej metody iteracyjnej zdefiniowanej w przestrzeni Kryłowa (metody gradientów sprzężonych). W pierwszej części pracy opisano szczegóły implementacji schematu iteracyjnego rozwiązywania dyskretnej postaci równania Poissona, uogólniając sformułowanie również do zagadnień przestrzennie trójwymiarowych. W drugiej części pracy skoncentrowano się przedstawieniu czasu wykorzystania procesora podczas wykonywania najbardziej czasochłonnych operacji algebry liniowej na macierzach rzadkich. Oceny poprawności formalnej jak też i wydajności obliczeniowej stworzonego kodu sekwencyjnego dokonano poprzez rozwiązanie trzech zagadnień testowych z wykorzystaniem dwóch komputerów o różnej konfiguracji sprzętowej.

Słowa kluczowe: Metody iteracyjne, Metoda różnic skończonych, Równanie Poissona, Wydajność kodu sekwencyjnego

ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2009
Informatyka – Zeszyt 4

Jerzy Krawczuk[1]

RANDOM TEST FOR TRADING SYSTEM

Abstract: Many traders use mechanical systems to trade. Such trading systems usually are tested on historical data. It shows good performance in the past. This paper describes the method for evaluating such trading systems based on the hypothesis that the tested system is trading randomly [1]. Under this null hypothesis we build a random system of the same characteristics as the tested one. The random system presents the same probabilities of taking a long position (betting on the price going up) and taking a short position (betting on the price going down) as tested system. The only distinguishing factor is the performance of the system. Simulations of the random system are run many times to create a performance distribution that is used to verify the null hypothesis. The test system in this paper trades the S&P500 futures from January 2003 until September 2008, taking always either long or short positions (always in the market) and reinvesting the profits.

Keywords: mechanical trades, random signal test

1. Introduction

Mechanical systems are widely used by the traders, and become more and more sophisticated as more powerful computers are available and more complex financial instruments can be traded. Also some systems that perform well in the past, become less efficient now and new systems need to be developed. Over 20 years ago, well known Turtle Trading System was very profitable, but in current market conditions it's not working so well.

The major advantage of using mechanical system over the human trading is eliminating the human emotions. There are many described and well known human behaviours that are destructive during trading [2]. Probably the most destructive is allowing losses to grow. People don't like loosing. They do not want to commit that they make bad decision by keeping losing position in their portfolios. But in the case of profit, they tend to close position very quickly, not allowing for bigger gains. Another advantage of mechanical system is to have fixed money management policy, which secure portfolio from too high risk. Available leverage is very high that could

[1] Faculty of Computer Science, Bialystok Technical University, Białystok

make a trader take too big risk. Mechanical systems also reduce the cost of money management and saving investors time, since they do not need to watch market all the time.

Many small investors as well as big hedge funds are using mechanical systems. Probably one of the most sophisticated systems is used by the Renaissance Technologies [3]. The company employs many specialists also with non-financial backgrounds, including mathematicians, physicists, astrophysicists and statisticians. Another famous hedge funds are Global Alpha fund by Goldman Sachs and The Highbridge Statistical Market Neutral Fund by J.P Morgan Chase.

Mechanical systems are developed and tested on historical data. Usually they have some parameters, that can be optimized for higher profit in the past. The result of the same system in the future is unknown. To maximize chances of having similar results is to make sure, that system works in the past:

- for many different market data
- for many time periods
- for the lowest possible system parameters

Additional test for randomness of the system is described in this paper.

2. Random signal test

This methodology described in [1] comparing the system performance with similar random system. Such random system trades the same contracts in the same time period, only difference is that it's deciding on long/short position randomly. If such system is created, it's run many times to produce the distribution of performance. From this distribution critical values are read for desired significance levels. If the system performance is greater then the critical value, we reject the hypothesis of random system trading. Let's c be the critical value, the hypothesis test will be:

- H0: System performance is not significant better then one achieved by the random trading. $Performance(TestedSystem) <= c$
- H1: System performance is significant better then random trading. $Performance(TestedSystem) > c$

As the system performance many different measures can be used. Most common and basic measures are based on total profit. This could be percent profit of the start capital - Return On Investment. In this article annual ROI is used.

$$annualROI = (endcapital/startcapital)^{1/years} - 1 \tag{1}$$

The profit measures could be also adjusted by some of the risk measures. For example Alex Strashny in [1] uses profit divided by three times maximum drawdown. One of the well known risk adjusted measure is Sharp Ratio [6]. Another custom performance measures could also be used in random test, for example: Time to recovery, Percent of Wining Trades. The second one is also tested in this article, it is calculated as the number of profitable trades divided by the total number of trades.

3. Description of tested system

Tested system was run on almost 6 years of historical data. It present a good total return of 1508.37% (fig.1), that gives in average 63.3% annual ROI (return on investment). System always keep either long or short position on S&P500 futures. Decision about changing position is made once per day at the market close. Position is changing very often giving the average holding time 2.3 days. Profit is reinvested. System is using leverage of 2:1. Signal either long or short is calculated based on probability

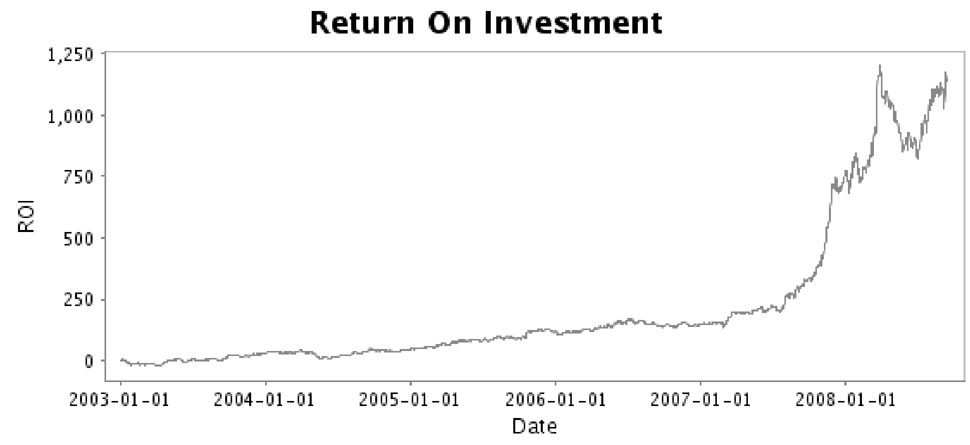


Fig. 1. Return on investment for tested system

distributions of short-term stock price movements. Distributions are calculated from over 5,000 US equity option prices. Then all of this information is used to derive the likely short-term direction of the S&P500. Actual transaction is made on futures contracts on the S&P500.

4. Creating the random system

Random system will be different from tested one only in a point of long/short signal. Tested system is using some rules that traders believe are working on the market giving the edge over other investors, while random system will do decision randomly. Other parameters of the systems stays the same, some of them are:

- transaction costs,
- size of a position,
- transaction time and contracts price, always closing prices
- time period
- used leverage

The random decision should also reflect characteristic of tested system. If system take long position it can keep it for a while as well as short position, that create the time series of positions:

$$LLLSLLSSSSSLLSSSLSLSLLL\ldots\ldots \tag{2}$$

when L stands for long position, and S for short. In this example system keeps long position for the first three days, then change for short for one day, and again for long for two days, and so on.

4.1 Bernoulli process [4]

We assume that every day decision is independent from the current position. Let's call L probability of the system signal saying be long, and according to our assumption $1-L$ is a probability of the system saying be short. The system final path probability could be wrote like this:

$$SystemProbability = L*L*L*(1-L)*L*L*(1-L)\ldots\ldots \tag{3}$$

Let say that the N is number of days on long position and M is number of days on short position, we can write previous equitation as:

$$SystemProbability = L^N*(1-L)^M \tag{4}$$

Using Maximum Likelihood method we assume that $SystemProbability$ is the highest of possible, and we estimate L as:

$$L = N/(N+M) \tag{5}$$

4.2 Markov process [5]

Let's say that the system is on long position, and it need to decide either to change short or stay long. Let's call $p(LS)$ probability of changing position from long to short, the probability of staying on long position will be $p(LL) = 1 - p(LS)$. Similar we will call $p(SL)$ probability of changing position from short to long, the probability of staying on short will be $p(SS) = 1 - p(SL)$. As an example let's consider that realized path is:

$$LLSLSS \tag{6}$$

Let's assume that probability of the first position is 0.5. Now we can describe probability of realized path 6 as:

$$SystemProbability = 0.5 * p(LL) * p(LS) * p(SL) * p(LS) * p(SS) \tag{7}$$

Let's define:
PL – number of days staying on long position
CL – number of days changing position from long to short
PS – number of days staying on short position
CS – number of days changing position from short to long

Please note that those numbers of days do not include first day. Now we can write the probability of any realized path as:

$$SystemProbability = 0.5 * p(LL)^{PL} * p(LS)^{CL} * p(SS)^{PS} * p(SL)^{CS} \tag{8}$$

Same as the function of 2 variables:

$$0.5 * (1 - p(LS))^{PL} * p(LS)CL * (1 - p(SL))^{PS} * p(SL)^{CS} \tag{9}$$

We are looking for $p(LS)$ and $p(SL)$ that will maximize probability of realized path. Let's notice that this function can be write as $f(p(LS)) * g(p(SL))$, both functions are non negative, so we can search for maximum of

$$(1 - p(LS))^{PL} - p(LS)^{CL} \tag{10}$$

and independently of

$$(1 - p(SL))^{PS} - p(SL)^{CS} \tag{11}$$

Using Maximum Likelihood method we estimate probabilities as:

$$p(LS) = CL/(CL + PL) \tag{12}$$

$$p(SL) = CS/(CS + PS) \tag{13}$$

5. Random System

Markov process was used to create random system since tested system tend to keep chosen position. That was reflected in average holding time of 2.3 days. Simulations was running from January 2nd 2003 until September 26th 2008. There was $PL = 427$ days keeping long position, $CL = 316$ days of changing position from long to short, $PS = 386$ days keeping short position, and $CS = 315$ changing from short to long.

Probabilities for tested system of changing position was calculated using 12 and 13:

- p(LS) = 0.4253
- p(SL) = 0.4487

Software that perform simulations was modified to trade based on calculated probabilities. At each day random number R is generated from unified distribution in range (0,1) and conditions for changing position are checked:

```
if system has long position then
    if $R < p(LS)$ then
        change to short;
    end
end
if system has short position then
    if $R < p(SL)$ then
        change to long;
    end
end
```

Algorithm 1: Random signal execution

1,000 random simulations was run produces distribution for ROI of random system showed in fig.2, since best result wasn't higher then one of the tested system another 10,000 simulations was run fig.3. In this case also the best result wasn't higher then achieved by the tested system. Second tested performance measure is Percent Of Wining Trades (POWD) fig.4.

6. Veryfication hypothesis of random trades

The level of significance is the probability of rejecting the null hypothesis when it is true. In simple case of just one system the critical value of 5% significance is just the 95th percentile of a performance distribution. In case of ROI and 10,000 random

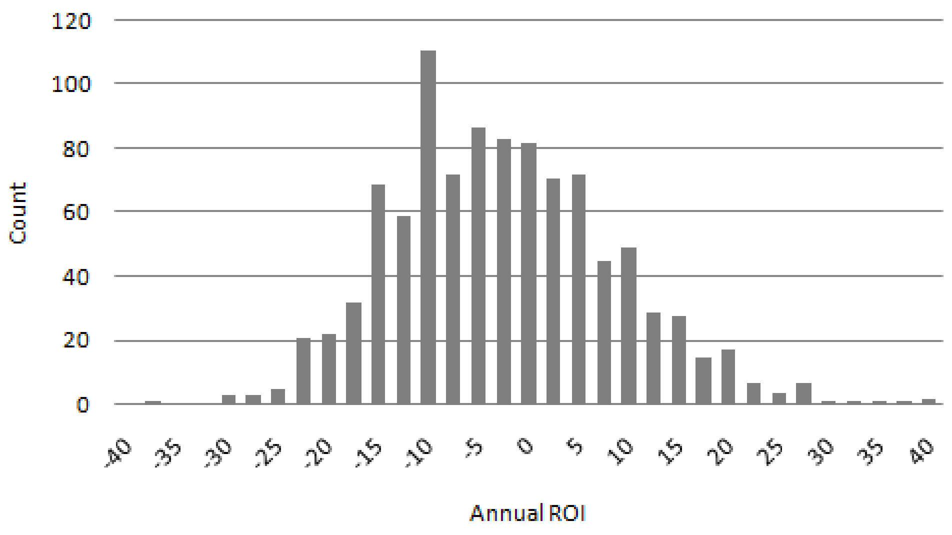


Fig. 2. ROI distribution for 1,000 random system simulations

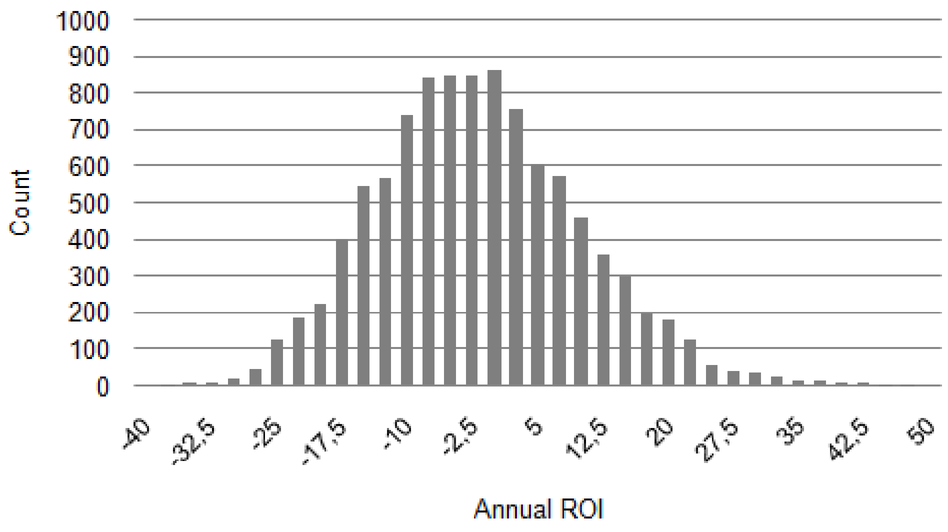


Fig. 3. ROI distribution for 10,000 random system simulations

simulations 95th percentile is 17.48 and 99th percentile is 26.63. For both significance levels we reject the null hypothesis since system annual ROI is 63.3%. Also

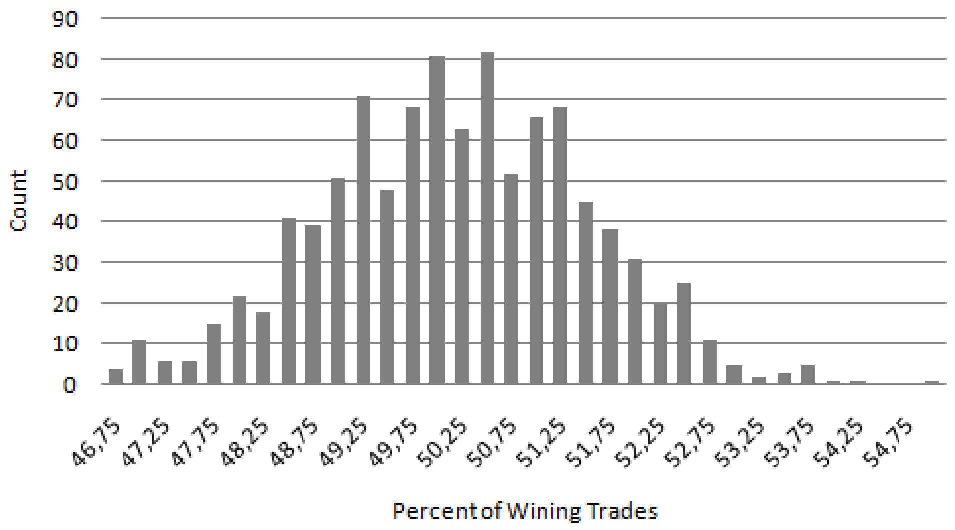


Fig. 4. POWD distribution for 1,000 random system simulations

best results in 10000 simulations was 50.30, so we can reject hypothesis of random trading even with lower significance level. In case of POWD and 1,000 random simulations 95th percentile is 52.27 and 99th percentile is 53.44 and the system Percent Of Winning Trades is 53.24%. In case of this performance measure we can reject null hypothesis only on 5% significance level, but we can not on 1%.

However, tested system got some parameters, and during optimizations best ones was picked up. For example 9 days moving average was used. But during optimizations also other values was tested like 8 and 10 days. If the system is not to much fitted to data, results for similar parameters should also produce good results. Additional tests for 8 and 10 days mean was performed. For each value probabilities p(LS) and p(SL) was calculated and 1,000 simulations was run to read the critical values for 1% and 5% significance levels. As shown in table 6. in all 3 cases hypothesis of random trading is rejected on both significance levels. Results for other values of moving average parameter are shown on fig.5.

7. Conclusion

This paper described the test of random trades for trading system. Long/short signals for random system was created using Markov process, other system properties stays the same. Null hypothesis of random signals was rejected for tested system on both

Table 1. Random test for different values of average moving parameter

moving avg	annual ROI	alfa=0.05	alfa=0.01	p(LS)	p(SL)
8	36.00	15.89	27.38	0.42445	0.42897
9	63.3	17.48	26.63	0.42472	0.44872
10	41.62	16.61	25.69	0.40608	0.44348

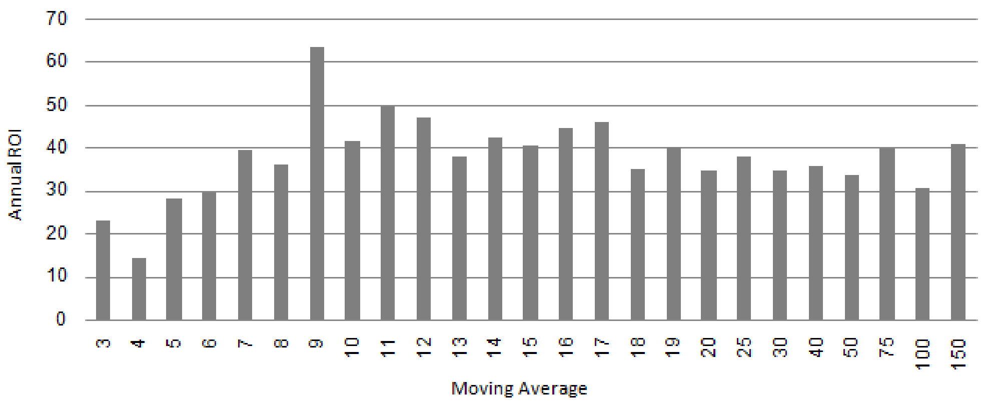


Fig. 5. Tested system results for different values of moving average parameter

1% and 5% significant levels for the ROI, and rejected only on 5% for the POWD performance measure. Is worth to note that any of 10,000 random simulations didn't outperform tested system's ROI . Since tested system goes through some optimizations to chose best parameters, tests for 2 additional moving average parameter was run. In this case also hypothesis of random signals was rejected.

Presented methodology could be the first step of verification trading system that was developed on historical data and show good past performance. It verify hypothesis that similar results could be achieved by the random signals. Even if system pass test for random signals, trader should also consider the ability of the system to work in the future, before using a system in real live. Things that should be considered are time period of the test, different market data and number of parameters used in the system.

References

[1] Alex Strashny: System evaluation based on past performance: Random Signals Test [http://ideas.repec.org/p/wpa/wuwpfi/0205003.html], Irvine, CA: University of California, May 2002.

[2] John R. Nofsinger: Psychology of Investing Second edition, Pearson Education 2005.
[3] Renaissance Technologies [http://en.wikipedia.org/wiki/Renaissance_Technologies].
[4] Carl W. Helstrom: Probability and Stochastic Processes for Engineers Macmillan Publishing Company 1984.
[5] Papoulis A.: Brownian Movement and Markoff Processes Ch. 15 in Probability, Random Variables, and Stochastic Processes McGraw-Hill 1984.
[6] Sharpe, William F.: Adjusting for Risk in Portfolio Performance Measurement Journal of Portfolio Management, Winter 1975, pp. 29-34.

TEST LOSOWOŚCI MECHANICZNYCH SYSTEMÓW TRANSAKCYJNYCH

Streszczenie Wielu inwestorów używa mechanicznych systemów transakcyjnych. Systemy takie testowane są na danych historycznych, gdzie osiągają dobre wyniki. W artykule tym opisano metodę testowania systemów transakcyjnych opartą na hipotezie iż system podejmuje decyzje losowo. Przy założeniu prawdziwości hipotezy konstruowany jest odpowiedni system losowy, który z takimi samymi prawdopodobieństwami generuje sygnały zajęcia pozycji. Opisywany system zajmuje pozycję długą (zakłada wzrost cen) bądź krótką (zakłada spadek cen) na kontraktach futures na indeks S&P500 (system zawsze posiada pozycję, nigdy nie jest poza rynkiem). Obliczenia dla systemu losowego wykonywane są wiele razy i tworzony jest rozkład prawdopodobieństwa wyników systemu. W oparciu o uzyskany rozkład weryfikowana jest hipoteza o losowości testowanego systemu.

Słowa kluczowe: mechaniczne systemy transakcyjne, test losowości

Artykuł zrealizowano w ramach pracy S/WI/2/08 Politechniki Białostockiej.

ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2009
Informatyka – Zeszyt 4

Małgorzata Krętowska[1]

PROGNOSTIC ABILITIES OF DIPOLES BASED ENSEMBLES – COMPARATIVE ANALYSIS

Abstract: In the paper, comparative analysis of ensembles of dipolar neural networks and regression trees was conducted. The techniques are based on the dipolar criterion function. Appropriate formation of dipoles (pairs of feature vectors) allows using them for analysis of censored survival data. As the result the methods return aggregated Kaplan-Meier survival function. The results, obtained by neural networks and regression trees based ensembles, are compared by using Brier score and direct and indirect measures of predictive accuracy.

Keywords: survival analysis, dipolar criterion, regression trees ensembles, neural networks ensembles

1. Introduction

In real data problems, the question about the future behavior of a given patient arises. Such situation is very common in survival data, in which the failure time is under investigation. In medical data, the failure is often defined as death or disease relapse and time is measured from the initial event, e.g. surgery. Analyzing the survival data requires taking into account censored observations, for which the exact failure time is unknown. The follow-up time for such patients gives us only the information, that the failure did not occur before.

Besides statistical techniques (the most common Cox's proportional hazards model [2]), which require some conditions to fulfill, other non-statistical methods are developed. Artificial neural networks and regression trees belong to the most popular ones. Recently, also methods concerning the use of ensemble of regression trees in prognosis of survival time appear. Their application allows receiving the tool unaffected by small changes in dataset, what is particularly important in discovering the risk factors. Hothorn *et al.* [5] proposes boosting survival trees to create aggregated survival function. Krętowska [11] developed the approach by using the dipolar regression trees or dipolar neural networks [12] instead of the structure proposed in

[1] Faculty of Computer Science, Białystok Technical University, Białystok

[5]. The technique proposed by Ridgeway [15] allows minimizing the partial likelihood function (boosting Cox's proportional hazard model). The Hothorn *et al.* [6] developed two approaches for censored data: random forest and gradient boosting.

The paper is organized as follows. In Section 2. the introduction to survival data as well as Kaplan-Meier survival function are presented. Section 3. introduces the idea of dipoles and in Section 4. two dipoles based structures: neural networks and regression trees are described. In Section 5. the algorithm of building the ensemble of predictors is presented. Experimental results are presented in Section 7. They were carried out on the base of two real datasets. The first one contains the feature vectors describing the patients with primary biliary cirrhosis of the liver [3], the other includes the information from the Veteran's Administration lung cancer study [7]. Section 8. summarizes the results.

2. Kaplan-Meier survival function

We have learning sample $L = (\mathbf{x}_i, t_i, \delta_i)$, $i = 1, 2, ..., n$, where $\mathbf{x}_i$ is N-dimensional covariates vector, t_i - survival time and δ_i - failure indicator, which is equal to 0 for censored cases and 1 for uncensored ones.

The distribution of random variable T, which represents the true survival time, may be described by the marginal probability of survival up to time $t > 0$ ($S(t) = P(T > t)$). The estimation of survival function $S(t)$ may be done by using the Kaplan-Meier product limit estimator [8], which is calculated on the base of learning sample L and is denoted by $\hat{S}(t)$:

$$\hat{S}(t) = \prod_{j|t_{(j)} \leq t} \left(\frac{m_j - d_j}{m_j} \right) \tag{1}$$

where $t_{(1)} < t_{(2)} < \ldots < t_{(D)}$ are distinct, ordered survival times from the learning sample L, in which the event of interest occurred, d_j is the number of events at time $t_{(j)}$ and m_j is the number of patients at risk at $t_{(j)}$ (i.e., the number of patients who are alive at $t_{(j)}$ or experience the event of interest at $t_{(j)}$).

The 'patients specific' survival probability function is given by $S(t|\mathbf{x}) = P(T > t|\mathbf{X} = \mathbf{x})$. The conditional survival probability function for the new patient with covariates vector $\mathbf{x}_{new}$ is denoted by $\hat{S}(t|\mathbf{x}_{new})$.

3. Dipoles

The methodology used during the learning process of artificial neural network and induction of regression tree bases on the concept of dipole [1]. The dipole is a pair of different covariate vectors $(\mathbf{x}_i, \mathbf{x}_j)$ from the learning set. Mixed and pure dipoles are distinguished. Mixed dipoles are formed between objects that should be separated, while pure ones between objects that are similar from the point of view of the analyzed criterion. The aim is to find such a hyper-plane $H(\mathbf{w}, \theta)$ that divides possibly high number of mixed dipoles and possibly low number of pure ones. It is done by minimization of the dipolar criterion function.

Two types of piece-wise linear and convex penalty functions $\varphi_j^+(\mathbf{v})$ and $\varphi_j^-(\mathbf{v})$ are considered:

$$\varphi_j^+(\mathbf{v}) = \begin{cases} \delta_j - < \mathbf{v}, \mathbf{y}_j > & \text{if} < \mathbf{v}, \mathbf{y}_j > \leq \delta_j \\ 0 & \text{if} < \mathbf{v}, \mathbf{y}_j > > \delta_j \end{cases} \tag{2}$$

$$\varphi_j^-(\mathbf{v}) = \begin{cases} \delta_j + < \mathbf{v}, \mathbf{y}_j > & \text{if} < \mathbf{v}, \mathbf{y}_j > \geq -\delta_j \\ 0 & \text{if} < \mathbf{v}, \mathbf{y}_j > < -\delta_j \end{cases} \tag{3}$$

where δ_j is a margin ($\delta_j = 1$), $\mathbf{y}_j = [1, x_1, \ldots, x_N]^T$ is an augmented covariate vector and $\mathbf{v} = [-\theta, w_1, \ldots, w_N]^T$ is an augmented weight vector. Each mixed dipole $(\mathbf{y}_i, \mathbf{y}_j)$, which should be divided, is associated with function $\varphi_{ij}^m(\mathbf{v})$ being a sum of two functions with opposite signs ($\varphi_{ij}^m(\mathbf{v}) = \varphi_j^+(\mathbf{v}) + \varphi_i^-(\mathbf{v})$ or $\varphi_{ij}^m(\mathbf{v}) = \varphi_j^-(\mathbf{v}) + \varphi_i^+(\mathbf{v})$). For pure dipoles that should remain undivided we associate function: $\varphi_{ij}^p(\mathbf{v})$ ($\varphi_{ij}^p(\mathbf{v}) = \varphi_j^+(\mathbf{v}) + \varphi_i^+(\mathbf{v})$ or $\varphi_{ij}^c(\mathbf{v}) = \varphi_j^-(\mathbf{v}) + \varphi_i^-(\mathbf{v})$). A dipolar criterion function is a sum of the penalty functions associated with each dipole:

$$\Psi_d(\mathbf{v}) = \sum_{(j,i) \in I_p} \alpha_{ij} \varphi_{ij}^p(\mathbf{v}) + \sum_{(j,i) \in I_m} \alpha_{ij} \varphi_{ij}^m(\mathbf{v}) \tag{4}$$

where α_{ij} determines relative importance (price) of the dipole $(\mathbf{y}_i, \mathbf{y}_j)$, I_p and I_m are the sets of pure and mixed dipoles, respectively.

The rules of dipoles formations depend on the purpose of our research. Assuming that the analysis aims at dividing the feature space into such areas, which would include the patients with similar survival times, pure dipoles are created between pairs of feature vectors, for which the difference of failure times is small, mixed dipoles - between pairs with distant failure times. Taking into account censored cases the following rules of dipole construction can be formulated:

1. a pair of feature vectors $(\mathbf{x}_i, \mathbf{x}_j)$ forms the pure dipole, if
   - $\sigma_i = \sigma_j = 1$ and $|t_i - t_j| < \eta$
2. a pair of feature vectors $(\mathbf{x}_i, \mathbf{x}_j)$ forms the mixed dipole, if
   - $\sigma_i = \sigma_j = 1$ and $|t_i - t_j| > \zeta$
   - ($\sigma_i = 0, \sigma_j = 1$ and $t_i - t_j > \zeta$) or ($\sigma_i = 1, \sigma_j = 0$ and $t_j - t_i > \zeta$)

Parameters η and ζ are equal to quartiles of absolute values of differences between uncensored survival times. The parameter η is fixed as 0.2 quartile and ζ - 0.6.

As the result of minimization of the dipolar criterion function we receive the values of parameters **v** of the hyper-plane. Depending on the set of dipoles, parameters **v** describe the neuron in artificial neural network or internal node in regression tree.

4. Individual prognostic structures

Two prognostic structures are considered in the paper: dipolar neural network [10] and regression tree [9]. The basic element of the structures (that is binary neurons and internal nodes) are characterized by the hyper-plane with parameters **v**:

$$z = f(\mathbf{y}, \mathbf{v}) = \begin{cases} 1 & \text{if } \mathbf{v}^T\mathbf{y} \geq 0 \\ 0 & \text{if } \mathbf{v}^T\mathbf{y} < 0 \end{cases} \tag{5}$$

From the geometrical point of view an element divides a feature space into two subspaces by using hyperplane $H(\mathbf{v}) = \{\mathbf{y} : \mathbf{v}^T\mathbf{y} = 0\}$. If the vector **y** is situated on the positive side of the hyper-plane, the element returns 1 ($z = 1$).

Neural network

A dipolar neural network model, considered in the paper, consists of two layers: input and output layer. The neurons weight values are obtained by sequential minimization of the dipolar criterion functions. The function is built from all the pure dipoles and those mixed dipoles which were not divided by previous neurons. The learning phase is finished when all the mixed dipoles are divided. The other, optimization phase, aims at distinguishing and enlargement of prototypes (i.e. active fields which contain the largest number of feature vectors **x**) and at reduction of redundant neurons [10].

The output layer of R binary neurons divided the N-dimensional feature space into disjoint regions - *active fields* (AF). Each region is represented by R-dimensional output vector: $\mathbf{z} = [z_1, z_2, \ldots, z_R]^T$, where $z_i \in \{0, 1\}$. As the result, the set of active fields $SAF = \{AF^1; AF^2; \ldots, AFi^k\}$ is received. Each active field AF^j contains the subset L^j of observations from the learning sample L.

Regression tree

Hierarchical and sequential structure of a regression tree recursively partitions the feature space. The tree consists of terminal nodes (leaves) and internal (non-terminal) nodes. An internal node contains a split (5), which tests the value of an expression of the covariates. Each distinct outcome (0 or 1) of the test generates one child node, which means that all non-terminal nodes have two child nodes. A terminal node generates no descendant. The function in a given node is designed on the base on those feature vectors that have reached the node. The induction of survival tree is stopped if one of the following conditions is fulfilled: 1) all the mixed dipoles are divided; 2) the set that reach the node consists of less than 5 uncensored cases.

Each terminal node represents one region in the N-dimensional feature space. Similarly to the neural network results, the leave j contains the subset L^j of observations from the learning sample L.

5. Ensembles of predictors

Let assume, that we have a set of k dipolar predictors (dipolar neural networks or dipolar regression trees): DP_i, $i = 1,2,\ldots,k$. The set is called ensemble when each of k predictors is generated on base of k learning samples $(L_1, L_2, \ldots, L_k)$ drawn with replacement from a given sample L. As the result of each dipolar predictor DP_i, the set $SL_i = \{L_i^1; L_i^2; \ldots, L_i^{k_i}\}$ of observations from learning sample L_i. Having a new covariate vector $\mathbf{x}_{new}$, each DP_i, $i = 1,2,\ldots,k$ returns the subset of observations $L_i(\mathbf{x}_{new})$ which is connected with region (or active field in case of neural networks), to which the new vector belongs. Having k sets $L_i(\mathbf{x}_{new})$, aggregated sample $L_A(\mathbf{x}_{new})$ is built [5]:

$$L_A(\mathbf{x}_{new}) = \{L_1(\mathbf{x}_{new}); L_2(\mathbf{x}_{new}); \ldots; L_k(\mathbf{x}_{new})\}$$

The aggregated conditional Kaplan-Meier survival function, calculated on the base of set $L_A(\mathbf{x}_{new})$ can be referred to as $\hat{S}_A(t|\mathbf{x}_{new})$.

The algorithm for receiving the aggregated survival function is as follows:

1. Draw k bootstrap samples $(L_1, L_2, \ldots, L_k)$ of size n with replacement from L
2. Induction of dipolar predictor DP_i based on each bootstrap sample L_i, $i = 1,2,\ldots,k$
3. Build aggregated sample $L_A(\mathbf{x}_{new}) = \{L_1(\mathbf{x}_{new}); L_2(\mathbf{x}_{new}), \ldots, L_k(\mathbf{x}_{new})\}$
4. Compute the Kaplan-Meier aggregated survival function for a new observation $\mathbf{x}_{new}$: $\hat{S}_A(t|\mathbf{x}_{new})$.

6. Measures of predictive accuracy

Beside the problems concerning the use of censored data in the process of building the prediction tool, the question how to evaluate the prediction ability of received models appears. The lack of exact failure times for a part of data causes that the classical measures based on difference between empirical and theoretical values can not be used. Instead of them, other, censoring oriented, measures are proposed.

One of them is the Brier score introduced by Graf *at al.* [4]. The Brier score as a function of time is defined by

$$\begin{aligned} BS(t) = \tfrac{1}{n}\textstyle\sum_{i=1}^{N}(\hat{S}(t|\mathbf{x}_i)^2 I(t_i \le t \wedge \sigma_i = 1)\hat{G}(t_i)^{-1} + \\ (1-\hat{S}(t|\mathbf{x}_i))^2 I(t_i > t)\hat{G}(t)^{-1}) \end{aligned} \qquad (6)$$

where $\hat{G}(t)$ denotes the Kaplan-Meier estimator of the censoring distribution. It is calculated on the base of observations $(t_i, 1-\delta_i)$. $I(condition)$ is equal to 1 if the condition is fulfilled, 0 otherwise. The BS equal to 0 means the best prediction.

The Brier score belongs to direct estimators of prediction ability, because it uses the information explicitly from the data. Another direct approach is proposed by Schemper and Henderson [14]. The predictive accuracy (without covariates), expressed by absolute predictive error (*APE*), at each distinct failure time $t_{(j)}$ is defined as:

$$\begin{aligned} \hat{M}(t_{(j)}) = \quad & \tfrac{1}{n}\textstyle\sum_{i=1}^{n}\left[I(t_i > t_{(j)})(1-\hat{S}(t_{(j)})) + \delta_i I(t_i \le t_{(j)})\hat{S}(t_{(j)}) + \right. \\ & \left.(1-\delta_i)I(t_i \le t_{(j)})\left\{(1-\hat{S}(t_{(j)}))\frac{\hat{S}(t_{(j)})}{\hat{S}(t_i)} + \hat{S}(t_{(j)})(1-\frac{\hat{S}(t_{(j)}}{\hat{S}(t_i)})\right\}\right] \end{aligned} \qquad (7)$$

The measure with covariates ($\hat{M}(t_{(j)}|\mathbf{x})$) is obtained by replacing $\hat{S}(t_{(j)})$ by $\hat{S}(t_{(j)}|\mathbf{x})$ and $\hat{S}(t_i)$ by $\hat{S}(t_i|\mathbf{x})$. To receive overall estimators of *APE* with ($\hat{D}_x$) and without covariates ($\hat{D}$) the weighed averages of estimators over failure times are calculated:

$$\hat{D} = w^{-1}\sum_j \hat{G}(t_{(j)})^{-1} d_j \hat{M}(t_{(j)}) \qquad (8)$$

$$\hat{D}_{\mathbf{x}} = w^{-1}\sum_j \hat{G}(t_{(j)})^{-1} d_j \hat{M}(t_{(j)}|\mathbf{x}) \qquad (9)$$

where $w = \sum_j \hat{G}(t_{(j)})^{-1} d_j$, d_j is the number of events at time $t_{(j)}$ and $\hat{G}(t)$ denotes the Kaplan-Meier estimator of the censoring distribution (see equation 6).

The indirect estimation of predictive accuracy was proposed by Schemper [13]. In the approach the estimates (without $\tilde{M}(t_{(j)})$ and with covariates $\tilde{M}(t_{(j)}|\mathbf{x})$) are defined by

$$\tilde{M}(t_{(j)}) = 2\hat{S}(t_{(j)})(1-\hat{S}(t_{(j)})) \qquad (10)$$

$$\tilde{M}(t_{(j)}|\mathbf{x}) = 2n^{-1}\sum_i \hat{S}(t_{(j)}|\mathbf{x}_i)(1-\hat{S}(t_{(j)})|\mathbf{x}_i) \qquad (11)$$

The overall estimators of predictive accuracy with ($\tilde{D}_{S,\mathbf{x}}$) and without ($\tilde{D}_S$) covariates are calculated similarly to the estimators $\hat{D}_{\mathbf{x}}$ and $\hat{D}$. The only change is replacing $\hat{M}(t_{(j)})$ and $\hat{M}(t_{(j)}|\mathbf{x})$ by $\tilde{M}(t_{(j)})$ and $\tilde{M}(t_{(j)}|\mathbf{x})$ respectively.

Based on the above overall estimators of absolute predictive error, explained variation can be defined as:

$$\tilde{V}_S = \frac{\tilde{D}_S - \tilde{D}_{S,\mathbf{x}}}{\tilde{D}_S} \tag{12}$$

and

$$\hat{V} = \frac{\hat{D} - \hat{D}_{\mathbf{x}}}{\hat{D}} \tag{13}$$

7. Experimental results

All the experiments were performed using the ensemble of 200 dipolar predictors *DP*. The measures of predictive accuracy were calculated on the base of learning sample *L*. To calculate the aggregated survival function for a given example **x** from the learning set *L*, only such DP_i ($i = 1,2,\dots,200$) were taken into consideration, for which **x** was not belonged to the learning set L_i (i.e. **x** did not participate in the learning process of the DP_i).

The analysis was conducted on the base on two datasets. The first one is from the Mayo Clinic trial in primary biliary cirrhosis (*PBC*) of the liver conducted between 1974 and 1984 [3]. 312 patients participated in the randomized trial. Survival time was taken as a number of days between registration and death, transplantation or study analysis time in July 1986. Patients are described by the following features: age(*AGE*), sex, presence of edema, logarithm of serum bilirubin [mg/dl] (*LOGBILL*), albumin [gm/dl] (*ALBUMIN*), logarithm of prothrombin time [seconds], histologic stage of disease. Dataset contains 60 per cent of censored observations.

In table 1, the results for *PBC* dataset are presented. Three different measures of predictive accuracy were calculated for three methods: Kaplan-Meier estimator, ensemble of DNN (dipolar neural network) and ensemble of DRT (dipolar regression tree). As we can see the absolute predictive error for K-M estimator (which is equivalent to the model without covariates) is equal to 0.37 and is higher than for other two methods (0.29 - indirect approach (0.26 - direct approach) for EDNN and 0.23(0.22) for EDRT). Comparing the results received for EDNN and EDRT we can noticed that for the model with all covariates as well as for model with only one feature the predictive measures are better for EDRT. Brier score for EDNN is equal to 0.17 and is bigger by 0.1 than Brier score for EDRT. In case of indirect and direct APE - explained variation for EDNN is smaller (0.22 (0.26)) than for EDRT -

0.39(0.41). Taking into account individual factors, the order of them is the same for both methods. The most important prognostic factor is logarithm of serum bilirubin for which the explained variation is equal to 0.25 (0.25) and 0.34 (0.35) for EDNN and EDRT respectively. The influence of age and albumin for prediction of survival probability is less important.

Table 1. Measures of predictive accuracy for *PBC* dataset

Model	*BS* (12years)	Indirect *APE*/ Explained variation	Direct *APE*/ Explained variation
K-M Estimator	0.23	0.37	0.37
Ensemble of DNN			
all covariates	0.17	0.29/0.22	0.27/0.26
AGE	0.22	0.36/0.036	0.36/0.038
LOGBILL	0.17	0.28/0.25	0.28/0.25
ALBUMIN	0.22	0.33/0.11	0.33/0.12
Ensemble of DRT			
all covariates	0.16	0.23/0.39	0.22/0.41
AGE	0.18	0.33/0.11	0.33/0.12
LOGBILL	0.16	0.24/0.34	0.24/0.35
ALBUMIN	0.18	0.28/0.24	0.28/0.24

The other analyzed data set contains the information from the Veteran's Administration (*VA*) lung cancer study [7]. In this trial, male patients with advanced inoperable tumors were randomized to either standard (69 subjects) or test chemotherapy (68 subjects). Only 9 subjects from 137 were censored. Information on cell type (0 - squamous, 1 - small, 2 - adeno, 3 - large) - CELL TYPE, prior therapy, performance status at baseline (Karnofsky rating - KPS), disease duration in months (TIME) and age in years at randomization (AGE), was available.

The measures of predictive accuracy for *VA lung cancer* data was shown in table 2. The unconditional absolute predictive error is 0.335. The ensemble of DNN, built on the base of all the covariates, reduces the error by 0.035 or 0.045 for indirect and direct approach respectively. The ensemble of DRT reduces the error by 0.145 and 0.185. As for *PBC* dataset case the results are better for EDRT, but the order of prognostic factors is the same. The most important prognostic factor is KPS (error equal to 0.3 (029) and 0.25(0.25), for EDNN and EDRT respectively). Explained variation is 11 (13) and 25 (25) per cent. Taking into account the EDNN, other variables have the marginal influence on the prediction of survival probability, but in case of EDRT also the cell type is quite important (explained variation equal to 13 (12) per cent).

Table 2. Measures of predictive accuracy for *VA lung cancer* data

Model	*BS* (100 days)	Indirect *APE*/ Explained variation	Direct *APE*/ Explained variation
K-M Estimator	0.24	0.335	0.335
Ensemble of DNN			
all covariates	0.18	0.3/0.11	0.29/0.14
AGE	0.24	0.32/0.034	0.33/0.013
CELL TYPE	0.24	0.33/0.002	0.33/0.006
KPS	0.19	0.3/0.11	0.29/0.13
TIME	0.24	0.33/0.003	0.33/0.003
Ensemble of DRT			
all covariates	0.09	0.22/0.35	0.18/0.46
AGE	0.2	0.3/0.09	0.3/0.1
CELL TYPE	0.19	0.29/0.13	0.3/0.12
KPS	0.18	0.25/0.25	0.25/0.25
TIME	0.22	0.31/0.07	0.31/0.07

8. Conclusions

In the paper, prognostic abilities of two ensemble of dipolar predictors (neural networks and regression trees) were compared. The prognostic ability of the models was verified by several measures, such as the Brier score and direct and indirect estimators of absolute predictive errors: $\tilde{D}_{S,x}$, $\hat{D}_x$. In all cases the measures were better for ensemble of dipolar regression trees then for ensemble of neural networks. For *VA lung cancer* data the explained variation was equal to 0.35 (0.46) for EDRT and 0.11 (0.14) (indirect (direct approach)) for EDNN. Similarly for *PBC* dataset, the explained variation received for EDRT - 0.39 (0.41) was greater than for EDNN - 0.22 (0.26). It worth noticed than two method distinguished the same risk factors. The feature that influence the survival the most is Karnofsky rating in case of *VA lung cancer* data and serum bilirubin for *PBC* dataset. The results suggest that the way of creating the consecutive hyper-planes in regression trees approach allows using the information from the given learning sample in the better manner.

References

[1] Bobrowski L., Krętowska M., Krętowski M.: Design of neural classifying networks by using dipolar criterions. Proc. of the Third Conference on Neural Networks and Their Applications, Kule, Poland (1997) 689–694

[2] Cox D.R.: Regression models and life tables (with discussion). Journal of the Royal Statistical Society B **34** (1972) 187–220

[3] Fleming T.R.: Harrington D.P., Counting Processes and Survival Analysis. John Wiley & Sons, Inc. (1991)

[4] Graf E., Schmoor C., Sauerbrei W., Schumacher M.: Assessment and comparison of prognostic classification schemes for survival data. Statistics in Medicine **18** (1999) 2529–2545

[5] Hothorn T., Lausen B., Benner A., Radespiel-Troger M.: Bagging survival trees. Statistics in medicine **23** (2004) 77–91

[6] Hothorn T., Buhlmann P., Dudoit S., Molinaro A. M., van der Laan M. J.: Survival ensembles. [URL http://www.bepress.com/ucbbiostat/paper174] U.C. Berkeley Division of Biostatistics Working Paper Series **174** (2005)

[7] Kalbfleisch J.D., Prentice R.L.: The statistical analysis of failure time data. John Wiley & Sons, New York (1980)

[8] Kaplan E.L., Meier P.: Nonparametric estimation from incomplete observations, Journal of the American Statistical Association **5** (1958) 457–481

[9] Krętowska M.: Dipolar regression trees in survival analysis. Biocybernetics and biomedical engineering **24** (3) (2004) 25–33

[10] Krętowska M., Bobrowski L.: Artificial neural networks in identifying areas with homogeneous survival time, L.Rutkowski et al. (Eds.): ICAISC 2004, LNAI 3070, (2004) 1008–1013

[11] Krętowska M.: Random forest of dipolar trees for survival prediction, L.Rutkowski et al. (Eds.): ICAISC 2006, LNAI 4029, (2006) 909–918

[12] Krętowska M.: Ensemble of Dipolar Neural Networks in Application to Survival Data, L.Rutkowski et al. (Eds.): ICAISC 2008, LNAI 5097, (2008) 78–88

[13] Schemper M.: Predictive accuracy and explained variation. Statistics in Medicine **22** (2003) 2299–2308

[14] Schemper M., Henderson R.: Predictive accuracy and explained variation in Cox regression. Biometrics **56** (2000) 249–255

[15] Ridgeway G.: The state of boosting. Computing Science and Statistics **31** (1999) 1722–1731

[16] Blake, C., Merz, C.: UCI Repository of machine learning databases `[http://www.ics.uci.edu/~mlearn/MLRepository.html]`, Irvine, CA: University of California, Department of Information and Computer Science, 1998.

[17] Duda O.R., Heart P.E., Stork D.G.: Pattern Classification, Second edition, John Wiley & Sons, 2001.

[18] Quinlan J.: Induction of decision trees, Machine Learning 1(1), 1986, pp. 81-106.

ZDOLNOŚCI PROGNOSTYCZNE KOMITETÓW BAZUJĄCYCH NA DIPPOLACH - ANALIZA PORÓWNAWCZA

Streszczenie W pracy przedstawiona została analiza porównawcza własności prognostycznych komitetów bazujących na sieciach neuronowych oraz drzewach regresyjnych. Tworzenie kolejnych sięć przestrzeni cech w obu metodach polega na minimalizacji odpowiednio skonstruowanego kryterium dipolowego. Do porównania metod wykorzystano indeks Brier'a oraz pośrednią i bezpośrednią miarę jakości predykcji. Eksperymenty wykonane zostały w oparciu o dwa rzeczywiste zbiory danych: pacjentów z pierwotną marskością żółciową wątroby oraz z rakiem płuc. W obu przypadkach wyniki otrzymane dla komitetu drzew regresyjnych były lepsze niż dla komitetu sieci neuronowych. Dotyczyło to zarówno badania jakości całego modelu, do którego wzięte zostały wszystkie dostępne w zbiorze cechy, jak też jakości prognostycznej pojedynczych cech. Natomiast uszeregowanie poszczególnych cech jako czynników ryzyka było podobne w obu metodach. Podsumowując można powiedzieć, że sposób podziału przestrzeni cech zaproponowany w drzewach regresyjnych w lepszy sposób wykorzystuje informacje zawarte w zbiorze uczącym.

Słowa kluczowe: analiza przeżyć, kryterium dipolowe, komitety drzew decyzyjnych, komitety sieci neuronowych

Artykuł zrealizowano w ramach pracy badawczej W/WI/4/08.

ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2009
Informatyka – Zeszyt 4

Tomasz Łukaszuk[1]

FEATURE SELECTION USING CPL CRITERION FUNCTIONS

Abstract: Dimensionality reduction of a feature set is a common preprocessing step used for pattern recognition and classification applications. It is particularly important when a small number of cases is represented in a highly dimensional feature space. The method of the feature selection based on minimisation of a special criterion function (convex and piecewise-linear - CPL) is considered in the article. A comparison of the experimental results of this method with the results of NIPS2003 Feature Selection Challenge participant's methods is also included.

Keywords: feature selection, CPL criterion function, NIPS2003 Feature Selection Challenge

1. Introduction

The feature selection is the technique, commonly used in machine learning, of selecting a subset of the most important features for building robust learning models. By removing most irrelevant and redundant features from the data, feature selection helps improve the performance of models constructed on the base of that data. In other words the feature selection means neglecting such measurements (features) which have no significant influence on the final decisions [4].

Dimensionality reduction is a preprocessing step commonly applied in pattern recognition and classification applications. It makes easier the next data analysis steps by alleviating the effect of the curse of dimensionality, enhancing generalization capability, speeding up learning process and improving model interpretability. Feature selection also helps people to acquire better understanding about their data by telling them which features are the important features.

The feature selection is particularly important when the data sets are composed of a small number of elements in a highly dimensional feature space. The situation when a small number of elements is represented in a highly dimensional feature space

[1] Faculty of Computer Science, Balystok Technical University, Białystok

(*long feature vectors*) usually leads to the linear separability of data sets [3]. The genomic data sets contain examples of the "long feature vectors".

This paper is engaged in the feature selection by minimization of a special convex and piece-wise linear (CPL) criterion function. The minimization process allows to calculate the parameters of hyperplane separated the learning sets and to find the best set of features ensured the linear separability of them at once. Moreover the goal of the paper is to make a comparison of described method expermiental results with the NIPS2003 Feature Selection Challenge participant's methods results.

2. Approaches to feature selection

Feature selection in substance is a task consists in removing irrelevant and redundant features from the initial data (features) set. Irrelevant and redundant features means features with no or minimal effect on later decisions.

There are two ways of selecting features set. One consists in making a ranking of features according to some criterion and select certain number of the best features. The other is to select a minimum subset of features without learning perfomance deterioration [6]. In the second way the quality of the whole subset is evaluated.

Important aspects connected with feature selection are models and search strategies. Typical models are filter, wrapper, and embedded. Filter methods use some own internal properties of the data to select features. Examples of the properties are feature dependence, entropy of distances between data points, redundancy. In the wrapper methods the feature selection is connected with the other data analysis technique, such as classification, clustering algorithm, regression. The accompand technicque helps with evaluation of the quality of selected features set. An embedded model of feature selection integreates the selection in model building. An example of such method is the decision tree induction algorithm. At each node a feature has to be selected. Basic search strategies applied in feature selection are forward, backward, floating, branch-and-bound and randomized strategies [6]. Besides there are a lot of modifications and improvements of them.

3. Linear separability of data sets and feature selection

Let us consider data represented as the feature vectors $\mathbf{x}_j[n] = [x_{j1},...,x_{jn}]^T$ $(j = 1,...,m)$ of the same dimensionality n or as points in the n-dimensional feature space $F[n]$. The components x_i of the vectors $\mathbf{x}_j[n]$ are called features. We are considering a situation, when the data can be a mixed (a qualitative-quantitative) type. Some components x_{ji} of the vectors $\mathbf{x}_j[n]$ can be the binary ($x_i \in \{0,1\}$) and others the real numbers ($x_i \in \mathbf{R}^1$).

Let us take into consideration two disjoined sets C^+ and C^- composed of m feature vectors $\mathbf{x}_j$:

$$C^+ \cap C^- = \emptyset \; . \tag{1}$$

For example vectors from the first set represent patients suffered from certain disease and vectors from the second one represent patients without the disease. The *positive set* C^+ contains m^+ vectors $\mathbf{x}_j$ and the *negative set* C^- contains m^- vectors ($m = m^+ + m^-$).

We are considering the separation of the sets C^+ and C^- by the hyperplane $H(\mathbf{w},\theta)$ in the feature space $F[n]$.

$$H(\mathbf{w},\theta) = \{\mathbf{x} : \langle \mathbf{w},\mathbf{x}\rangle = \theta\} \tag{2}$$

where $\mathbf{w} = [w_1,...,w_n]^T \in \mathbf{R}^n$ is the weight vector, $\theta \in \mathbf{R}^1$ is the threshold, and $\langle \mathbf{w},\mathbf{x}\rangle$ is the inner product.

Definition 1. *The feature vector* $\mathbf{x}$ *is situated on the* positive side *of the hyperplane* $H(\mathbf{w},\theta)$ *if and only if* $\langle \mathbf{w},\mathbf{x}_j\rangle > \theta$ *and the vector* $\mathbf{x}$ *is situated on the* negative side *of* $H(\mathbf{w},\theta)$ *if and only if* $\langle \mathbf{w},\mathbf{x}_j\rangle < \theta$.

Definition 2. *The sets* C^+ *and* C^- *are* linearly separable *if and only if they can be fully separated by some hyperplane* $H(\mathbf{w},\theta)$ *(2):*

$$(\exists \mathbf{w},\theta) \quad (\forall \mathbf{x}_j \in C^+)\langle \mathbf{w},\mathbf{x}_j\rangle > \theta \quad \textbf{\textit{and}} \quad (\forall \mathbf{x}_j \in C^-)\langle \mathbf{w},\mathbf{x}_j\rangle < \theta \; . \tag{3}$$

In accordance with the relation (3), all the vectors $\mathbf{x}_j$ belonging to the set C^+ are situated on the positive side of the hyperplane $H(\mathbf{w},\theta)$ (2) and all the feature vectors $\mathbf{x}_j$ from the set C^- are situated on the negative side of this hyperplane.

The feature selection can be linked with searching of hyperplane $H(\mathbf{w},\theta)$ (2) separated the sets C^+ and C^-. It is possible to find a hyperplane in reduced feature space $F[n']$ $n' \leq n$. This fact results from the characteristic of the linear independence of the feature vectors $\mathbf{x}_j$ constituting the sets C^+ and C^- [2].

4. Convex and piece-wise linear (*CPL*) criterion function $\Phi_\lambda(\mathbf{w},\theta)$

If the sets C^+ and C^- are linearly separable there are very many hyperplanes $H(\mathbf{w},\theta)$ (2) divided them [3]. In order to avoiding overfitting of the model and obtainig good

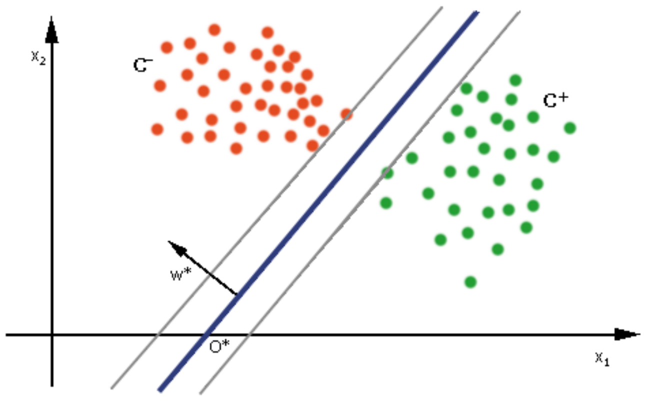


Fig. 1. Optimal hyperplane $H(\mathbf{w}^*, \theta^*)$ ensured the widest margin between itself and the elements of the sets C^+ and C^-

its generalization ability the optimal hyperplane $H(\mathbf{w}^*, \theta^*)$ should be found. Optimal hyperplane means the hyperplane ensured the widest margin between itself and the elements of the sets C^+ and C^-.

The hyperplane $H(\mathbf{w}^*, \theta^*)$ could be calculated by the minimization of the criterion function $\Phi_\lambda(\mathbf{w}, \theta)$ [3].

$$\Phi_\lambda(\mathbf{w},\theta) = \sum_{\mathbf{x}_j \in C^+} \alpha_j \varphi_j^+(\mathbf{w},\theta) + \sum_{\mathbf{x}_j \in C^-} \alpha_j \varphi_j^-(\mathbf{w},\theta) + \lambda \sum_{i \in I} \gamma_i \phi_i(\mathbf{w},\theta) \tag{4}$$

where $\alpha_j \geq 0$, $\lambda \geq 0$, $\gamma_i > 0$, $I = \{1, ..., n+1\}$.
The nonnegative parameters α_j determine relative importance (*price*) of particular feature vectors $\mathbf{x}_j$. The parameters γ_i. represent the *costs* of particular features x_i.

The function $\Phi_\lambda(\mathbf{w}, \theta)$ is the sum of the penalty functions $\varphi_j^+(\mathbf{w}, \theta)$ or $\varphi_j^-(\mathbf{w}, \theta)$ and $\phi_i(\mathbf{w}, \theta)$. The functions $\varphi_j^+(\mathbf{w}, \theta)$ are defined on the feature vectors $\mathbf{x}_j$ from the set C^+. Similarly $\varphi_j^-(\mathbf{w}, \theta)$ are based on the elements $\mathbf{x}_j$ of the set C^-.

$$(\forall \mathbf{x}_j \in C^+) \quad \varphi_j^+(\mathbf{w},\theta) = \begin{cases} 1+\theta-\langle \mathbf{w}, \mathbf{x}_j \rangle & if\ \langle \mathbf{w}, \mathbf{x}_j \rangle < 1+\theta \\ 0 & if\ \ \langle \mathbf{w}, \mathbf{x}_j \rangle \geq 1+\theta \end{cases} \tag{5}$$

and

$$(\forall \mathbf{x}_j \in C^-) \quad \varphi_j^-(\mathbf{w},\theta) = \begin{cases} 1+\theta+\langle \mathbf{w}, \mathbf{x}_j \rangle & if\ \langle \mathbf{w}, \mathbf{x}_j \rangle > -1+\theta \\ 0 & if\ \ \langle \mathbf{w}, \mathbf{x}_j \rangle \leq -1+\theta \end{cases} \tag{6}$$

The penalty functions $\phi_i(\mathbf{w}, \theta)$ are related to particular features x_i.

$$\phi_i(\mathbf{w},\theta) = \begin{cases} |w_i| & if\ 1 \leq i \leq n \\ |\theta| & if\ \ i = n+1 \end{cases} \tag{7}$$

The criterion function $\Phi_\lambda(\mathbf{w},\theta)$ (4) is the convex and piecewise linear (*CPL*) function as the sum of the *CPL* penalty functions $\varphi_j^+(\mathbf{w},\theta)$ (6), $\varphi_j^-(\mathbf{w},\theta)$ (7) and $\phi_i(\mathbf{w},\theta)$ (7). The basis exchange algorithm allows to find the minimum efficiently, even in the case of large multidimensional data sets C^+ and C^- [1].

$$\Phi_\lambda^* = \Phi_\lambda(\mathbf{w}^*,\theta^*) = min\,\Phi_\lambda(\mathbf{w},\theta) \geq 0 \tag{8}$$

The vector of parameters w^* and the parameter θ^* define the hyperplane $H(\mathbf{w}^*,\theta^*)$. It is the best hyperplane separated the sets C^+ and C^- according to the interpretation showed on the figure 1.

5. NIPS2003 Feature Selection Challenge

NIPS is the acronym of Neural Information Processing Systems. It is the annual conference name taken place in Vancouver, Canada from 1987. Its topics span a wide range of subjects including neuroscience, learning algorithms and theory, bioinformatics, image processing, and data mining [7].

In 2003 within the framework of NIPS Conference took place the challenge in feature selection. The organizers provided participants with five datasets from different application domains and called for classification results using a minimal number of features. All datasets are two-class classification problems. The data were split into three subsets: a training set, a validation set, and a test set. All three subsets were made available at the beginning of the benchmark, on September 8, 2003. The class labels for the validation set and the test set were withheld. The identity of the datasets and of the features (some of which, called probes, were random features artificially generated) were kept secret. The participants could submit prediction results on the validation set and get their performance results and ranking on-line for a period of 12 weeks. On December 1st, 2003, the participants had to turn in their results on the test set. The validation set labels were released at that time. On December 8th, 2003, the participants could make submissions of test set predictions, after having trained on both the training and the validation set [8]. Performance was assessed using several metrics, such as:

- the balanced error rate (the average of the error rate of the positive class and the error rate of the negative class),
- area under the ROC curve (the ROC curve is obtained by varying a threshold on the discriminant values (outputs) of the classifier, The curve represents the fraction of true positive as a function of the fraction of false negative),
- fraction of features selected,

– fraction of probes (random artificially generated features) found in the feature set selected.

The NIPS 2003 challenge on feature selection is over, but the website of the challenge is still open for post-challenge submissions. One can compare results by his own method with the challenge participant's methods results.

6. Numerical experiments

There was performed the experiments with the use of earlier described feature selection methods. The input data were taken from the NIPS2003 Feature Selection Challenge web site [8]. The author's own implementation of the feature selection methods was applied during experiments. The results were formated according to directions of the challenge organizers and compared with the results of challange participans.

6.1 Data sets

There are five datasets spaned a variety of domains (cancer prediction from mass-spectrometry data, handwritten digit recognition, text classification, and prediction of molecular activity). One dataset is artificial. All problems are two-class classification problems. For the purpose of challange the data sets were prepared appropriately. Preparing the data included, amoung other things:

– preprocessing data to obtain features in the same numerical range (0 to 999 for continuous data and 0/1 for binary data),
– adding „random" features (probes) distributed similarly to the real features in order to rank algorithms according to their ability to filter out irrelevant features,
– splitting the data into training, validation and test set.

6.2 Course of experiments

Large data dimensions and connected with it a big size of data files determine using a special methods of dealing with data and strong enough computer system. The author's own implementation of the feature selection method relied on the minimization of CPL criterion function (described in section 4.) with the use of the basis exchange algorithm [1] was applied. The calculation executed on the computer equiped with 64 bit linux operation system, Intel Core2 Quad processor and 4MB RAM memory.

Table 1. The data sets of NIPS2003 Feature Selection Challenge

Dataset	**Domain**	**Type**	**Features**	**Probes**	**Training exam.**	**Validat. exam.**	**Test exam.**
Ascene	mass-spectrometric data, patients with cancer (ovarian or prostate cancer), and healthy	Dense	10000	3000	100	100	700
Gisette	handwritten digits: the four and the nine	Dense	5000	2500	6000	1000	6500
Dexter	texts about "corporate acquisitions"	Sparse integer	20000	10053	300	300	2000
Dorothea	discovery drugs, predict which compounds bind to Thrombin	Sparse binary	100000	50000	800	350	800
Madelon	artificial data, XOR problem	Dense	500	480	2000	600	1800

The learning data sets were constructed from the objects from training and validation sets. It means the author as a participant of the challenge starts from the second challange stage, when labels of the object from validation set are known.

The applied feature selection method generates indexes of selected features and coefficients of the hyperplane separated the learnig sets C^+ and C^- (1) corresponding to the indexes. The obtained results needed to additional process in order to submit them via NIPS2003 challenge web site and compare author's methods with the challenge entries. The results on each dataset should be formatted in ASCII files. In the separated files should be placed the classifier outputs for training, validation and test examples, the forth file should include a list of feature indexes used for classification. The author's results were formated according to the instructions.

6.3 Results

Table 2 includes the outcomes of applying author's feature selection method with the data sets of NIPS2003 Feature Selection Challenge. The numbers were received from the challenge web site after submitting formated results.

The series of results consist of the Balanced Error and Area Under Curve values (described in section 5.) defined seperately for train, validation and test sets, the number of features used by classifier and its proportion to the whole set of features and also the number of artificial features (probes) in the selected features set and its proportion to the number of all features selected by method. Besides the results concerning particular data sets, the table 2 includes the average results for all five data sets in the last row.

A perfect feature selection method should characterized by as small as possible values of Balanced Error with reference to all part of data set: train, validation and test

Table 2. Results of author's method in the NIPS2003 challenge

Dataset	Balanced Error			Area Under Curve			Features		Probes	
	Train	**Valid**	**Test**	**Train**	**Valid**	**Test**	**#**	**%**	**#**	**%**
arcene	0.0000	0.0000	0.3084	1.0000	1.0000	0.6916	32	0.32	18	56.25
gisette	0.0000	0.0000	0.0571	1.0000	1.0000	0.9429	222	4.44	160	72.07
dexter	0.0000	0.0000	0.1560	1.0000	1.0000	0.8440	56	0.28	12	21.43
dorothea	0.0000	0.0000	0.3401	1.0000	1.0000	0.6599	44	0.04	33	75.00
madelon	0.2665	0.2517	0.4744	0.7335	0.7483	0.5256	500	100.00	480	96.00
overall	0.0533	0.0503	0.2672	0.9467	0.9497	0.7328		21.02		64.15

(and connected with it close to 1 value of Area Under Curve). The number of features selected by a perfect feature selection method does not defined and it depends on a type of data set. Nevertheless the number should not be excessive numerous. In case of artificial features a perfect feature selection method should not choose them. It means the number of them should be as small as possible or equal to 0 in perfect situation.

The table of results shows the author's feature selection mehtod (especially its applied version) is not a perfect one. Particularly bad results were produced in case of madelon data set. Moreover the method selected too many arificial features. It concerns all data sets. The number of features used in classification are in the adequate level (except madelon data set). In case of values of Balanced Error and Area Under Curve with reference to train and validatin data sets the method turned out to be a perfect one (except madelon data set).

6.4 Conclusions

The applied method of feature selection uses the linear classifier. The properties of that kind of classifier disqualify it as a good one with xor problem. So disadvantageous results for madelon dataset follow from the character of used classifier, because medalon dataset represents exactly xor problem.

On the basis of the results the applied method is found as the method with a tendency to overfitting. The classification errors do not occure in case of train and validation data sets, whereas in the test data sets the errors are equal about 25%. The substantial number of the artificial features in the selected features set points at the tendency to overfitting, too.

6.5 Comparison with NIPS2003 Feature Selection Challenge participant's results

According to rating placed on the web site [8] and created on the basis of the results provided by the participants of NIPS2003 challenge, the author's method has placed itself on 185th position (when classification criterion is the average Balanced Error on the test set). It means about a half of the list.

A large negative influence on the rating position has very bad outcome obtained with madelon data set. The applied method does not manage with that kind of data from its nature. If the results of madelon dataset were not taken into account and the average value based on the remaining four datasets, the author's method would achieve Balanced Error value refered to the test set equal 0.2154. It would improve the rating position of method to 154th position.

The author's method occupies 134th position if the proportion of the number of features used by classifier to the number of all features is the rating criterion. The result attained for madelon dataset has again a disadvantageous influence to the situation. If the results of madelon dataset were not taken into account, the method would rate on the high enought 40th position.

The bast methods employed by participants of the challenge have obtained the Balanced Error on the test set less than 7%. However a different methods have been used with particular data sets by a single participant as a rule. For example one of the competitor describes his method in the following manner: "Combination of Bayesian neural networks and classification based on Bayesian clustering with a Dirichlet diffusion tree model. A Dirichlet diffusion tree method is used for Arcene. Bayesian neural networks (as in BayesNN-large) are used for Gisette, Dexter, and Dorothea. For Madelon, the class probabilities from a Bayesian neural network and from a Dirichlet diffusion tree method are averaged, then thresholded to produce predictions." [8]. The other participants with well results applied among other things random forests method, SVM, a different form of neural networks.

In sum it could be ascertained that the results of author's method are not extremely well. Nevertheless they are not bad, too.

7. Concluding remarks and future work

The paper presents basic assumptions of the method of feature selection based on the minimisation of the CPL criterion function. It contains the results obtained from applying of described method with the data provided by the NIPS2003 Feature Selection Challenge organizers. In comparison with participants of the challenge the author's method has placed itself in the middle of list, particularly in case of the Balanced Error for test set, the most important rating criterion.

It needs to be noticed described experiment was the first experience of author with NIPS2003 challenge. On the basis of observations of the results list it can be stated that the approach to the challenge several times is the rule among the challenge participants. On every next approach the participant attains better rating position as the effect of improving own method. The purpose of author is also the improving his feature selection method and reaching higher rating position in the future.

The NIPS2003 Feature Selection Challenge is a good benchmark allowed on a competent estimating of the efficiency of improvements introduced in own feature selection methods as well as a comparising of own solutions with other from the research domain.

References

[1] Bobrowski L.: Design of Piecewise Linear Classifiers from Formal Neurons by Some Basis Exchange Technique, pp. 863–870 in: Pattern Recognition, 24(9), 1991.

[2] Bobrowski L., Łukaszuk T.: Selection of the linearly separable feature subsets, pp.544-549 in: Artificial intelligence and soft computing: ICAISC'2004, eds. Leszek Rutkowski, Jörg Siekmann, Ryszard Tadusiewicz, Lotfi A. Zadeh, Lecture Notes in Computer Science, vol.3070, 2004.

[3] Bobrowski L.: Eksploracja danych oparta na wypukłych i odcinkowo-liniowych funkcjach kryterialnych, Wyd. Politechniki Białostockiej, Białystok, 2005.

[4] Duda O.R., Heart P.E., Stork D.G.: Pattern Classification, Second edition, John Wiley & Sons, 2001.

[5] Fukunaga K.: Statistical Pattern Recognition, Academic Press, Inc., San Diego, 1990.

[6] Liu H., Motoda H.: Computational methods of feature selection, Chapman & Hall/CRC data mining and knowledge discovery series, Chapman & Hall/CRC, 2008.

[7] http://nips.cc/

[8] http://www.nipsfsc.ecs.soton.ac.uk/

SELEKCJA CECH Z WYKORZYSTANIEM FUNKCJI KRYTERIALNYCH TYPU CPL

Streszczenie Redukcja wymiarowości zbioru cech jest często używanym wstępnym krokiem przetwarzania danych stosowanym przy rozpoznawaniu wzorców i klasyfikacji. Jest ona szczególnie istotna kiedy mała liczba obserwacji jest reprezentowana w wysoko wymiarowej przestrzeni cech. W artykule rozważana jest metoda selekcji cech opierająca się na minimalizacji specjalnej funkcji kryterialnej (wypukłej i odcinkowo-liniowej - CPL). Załączono także porównanie wyników eksperymentów uzyskanych za pomocą opisanej metody z wynikami metod uczestników konkursu NIPS2003 Feature Selection Challenge.

Słowa kluczowe: selekcja cech, funckcja kryterialna typu CPL, konkurs NIPS2003 Feature Selection Challenge

Artykuł zrealizowano częściowo w ramach grantu MNiSW 3T11F01130 i w ramach pracy badawczej S/WI/2/08 Politechniki Białostockiej.

ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2009
Informatyka – Zeszyt 4

Agnieszka Makarec[1]

PROBABILISTIC AND NON-DETERMINISTIC SEMANTICS FOR ITERATIVE PROGRAMS

Abstract: In this paper probabilistic and non-deterministic programs are considered on the ground of logic of programs. We are interested in dependencies between nondeterministic and probabilistic interpretation of a program. The formal definitions of probabilistic and non-deterministic semantics are the starting point for our considerations. The emphasis is on differences in expressibility the halting property in probabilistic and non-deterministic logic of programs.

Keywords: non-deterministic program, probabilistic program, non-deterministic computation, probabilistic computation

1. Introduction

There are many reasons that make the problem of non-deterministic and probabilistic constructions an important field of study: average case analysis, randomized algorithms, distributed systems, concurrency, reliability, security. There were proposed several approaches to model of non-deterministic and random choice of the control in programming languages (cf. [2,3,4,6,7]). Non-determinism can be used to abstract over probabilities. It is a valuable tool that helps us understand the expressiveness of programming language constructs. However, the input to a program and the output may be described in terms of a probability distribution, in which case the average case behavior of the program can be formally analyzed.

2. Non-deterministic and probabilistic interpretations of random assignments

The considerations of the presented paper will be formulated in terms of logics of programs. We shall compare properties of iterative programs expressible in non-deterministic and probabilistic versions of algorithmic logic (cf. [1,8]). To compare these two approaches we assume that computations of probabilistic and non-deterministic programs are realized in a fixed algebraic structure for language L_{alg},

[1] Faculty of Computer Science, Bialystok Technical University, Białystok

containing assignments of the form $x := ?$ and the construction *either ...or ...* . Probabilistic and non-deterministic interpretation of assignments $x := ?$ should be *consistent* in a natural sense: an element is probabilistically generated with a positive probability if and only if non-deterministic generation of this element is possible.

The concepts of them seem to be essentially different but some dependencies can be formulated and proved. Namely, if in probabilistic interpretation, program K halts with positive probability, then in the non-deterministic interpretation the formula expressing the possibility of halting is valid. In the present paper we restrict ourselves to finite interpretations because of the efficiency of verification of programs properties.

The algorithmic language L_{alg}, being an extension of first-order language L, is an abstract programming language of iterative programs.

There are three kinds of well-formed expressions in L_{alg}: terms, formulas and programs. The alphabet of the language L_{alg} contains enumerable sets of predicates Ψ, functors Φ and a countable set V_1 of individual variables. The set T of terms and the set F of first-order formulas are defined as in the classical logic. The set Π of iterative programs is the least set of expressions containing assignments $x := ?$, $x := \tau$, where $x \in V_1, \tau \in V_1$, and closed under the program constructions: *begin* M; N *end* , *if* γ *then* M *else* N , *while* γ *do* M, *either* M *or* N.

3. Probabilistic Algorithmic Logic

In this subsection we describe the syntax and semantics of language L_{prob}, which is an extension of L_{alg} (such that $L \subset L_{alg} \subset L_{prob}$). The syntax and semantics of language L_{prob} derives from Probabilistic Algorithmic Logic PrAL [1].

3.1 Syntax of L_{prob}

The alphabet of the language L_{prob} contains:
Ψ – an enumerable set of predicates (the equality sign = is distinguished), which includes subset Ψ_R of arithmetical predicates and Ψ_1 of non arithmetical predicates,
Φ – an enumerable set of functors, which includes subset $\Phi_R = \{+,-,*,/\}$ of arithmetical functors and subset Φ_1 of non arithmetical functors,
V – an countable set of variables, which includes subset V_1 of individual variables and subset V_R of real variables,
C – a finite set of real constants.
L_{prob} is a two-sorted language with the sets of terms T and T_R , and the set of all formulas F_{prob}.
The set T_R of arithmetical terms is the least set of expressions such that:

i. if $\alpha \in F$ and K$\in \Pi$ then $P(\alpha), P(\mathrm{K}\alpha) \in T_R$,
ii. $V_R \subset T_R$,
iii. if $\varphi \in \Phi_R$ is an j-argument functor and $\tau_1, \ldots, \tau_j \in T_R$ then $\varphi(\tau_1, \ldots, \tau_j) \in T_R$

The set F_{prob} is the least set of expressions such that:

i. if $\alpha \in F$ and K$\in \Pi$ then $\mathrm{K}\alpha \in F_{prob}$,
ii. if $\alpha, \beta \in F_{prob}$ then $\alpha \vee \beta, \alpha \wedge \beta, \alpha \Rightarrow \beta, \neg\alpha \in F_{prob}$,
iii. if $\xi \in \Psi_R$ is an j-argument predicate and $\tau_1, \ldots, \tau_j \in T_R$ then $\xi(\tau_1, \ldots, \tau_j) \in F_{prob}$.

The set F_0 of *open formulas* is the least set of expressions such that:

i. if $\sigma \in \Psi_1$ is an j-argument predicate and $\tau_1, \ldots, \tau_j \in T$ then $\sigma(\tau_1, \ldots, \tau_j) \in F_0$,
ii. if $\alpha, \beta \in F_0$ then $\alpha \vee \beta, \alpha \wedge \beta, \alpha \Rightarrow \beta, \neg\alpha \in F_0$.

3.2 Semantics of L_{prob}

The interpretation of the language L_{prob} will be understood as the triple $\langle \Im, \rho, p \rangle$, where $\Im$ is a structure for L with the universe $U = \{u_1, \ldots, u_r\}$ and $\rho : U \to [0,1]$ is the fixed probabilistic distribution on the set U connected with a random assignment $x := ?$, such that

$$\rho(u_i) = p_i, \sum_{i=1,\ldots,r} p_i = 1. \tag{1}$$

The number p, satisfying $0 < p < 1$, corresponds to the probability of choosing the subprogram M_1 as the result of the realization of the construction *either* M_1 *or* M_2.

By a *valuation* of individual variables we mean a mapping $\omega : V_1 \to U$, where $V_1 = \{x_1, x_2, \ldots\}$. By a valuation of real variables we mean a mapping $\omega_R : V_R \to R$. Let us denote by $\mathrm{K}(x_1, \ldots, x_n)$ the probabilistic program with all variables among $\{x_1, \ldots, x_n\}$. We will write it simply K when no confusion can arise. Each *state* of program K is described by the valuation $\omega = (\omega(x_1), \ldots, \omega(x_n))$. We will denote by $W = \{\omega_1, \ldots, \omega_l\}, l = r^n$ the set of all possible valuations of variables from $\{x_1, \ldots, x_n\}$. Fact that the valuation ω in the structure $\langle \Im, \rho, p \rangle$ satisfies the formula α will be denoted by $\langle \Im, \rho, p \rangle, \omega \models \alpha$ (or shortly $\omega \models \alpha$).

The interpretation of open formulas and terms in the language L_{prob} is classical and it was presented in [1,5].
The element $\tau_{\langle \Im, \rho, p \rangle}(\omega)$ of U is called *the value of the term* τ *in the structure* $\langle \Im, \rho, p \rangle$ *at the valuation* ω and it will be shortly denoted by $\tau(\omega)$ (when no confusion can arise).

3.3 Computational semantics of L_{prob}

By a *configuration* in the structure $\langle \Im, \rho, p \rangle$ we understand any ordered triple of the form $s = \langle \omega; q; \overrightarrow{K} \rangle$, where ω is a valuation, q is a probability of occurring that valuation, $\overrightarrow{K} = (\mathrm{K}_1, \ldots, \mathrm{K}_m)$ is a finite sequence of programs, which are executed in order as they are written (the sequence may be empty).

A *computation* of a probabilistic program K with an input valuation ω, which appears with probability q, we mean as the sequence of configurations satisfying the following conditions.

I. The first element of this sequence is in the form of $s_1 = \langle \omega; q; \mathrm{K} \rangle$.

II. If i-th configuration $s_i = \langle \omega'; q'; \mathrm{K}_1, \ldots, \mathrm{K}_m \rangle$ of the sequence is determined and q' denotes a probability of occurring valuation ω' then s_{i+1} is the next configuration and:

   i. If K_1 is in the form of $x := \tau$, then $s_{i+1} = \langle \omega''; q'; \mathrm{K}_2, \ldots, \mathrm{K}_m \rangle$, where

$$\omega''(y) = \begin{cases} \tau(\omega') \text{ iff } y = x, \\ \omega'(y) \text{ iff } y \neq x. \end{cases} \tag{2}$$

   ii. If K_1 is in the form of $x :=?$, then $s_{i+1} = \langle \omega''; q' \cdot \rho(u); \mathrm{K}_2, \ldots, \mathrm{K}_m \rangle$, where $\rho(u)$ denotes probability assignment of an element $u \in U$ to a variable x and

$$\omega''(y) = \begin{cases} u \quad\;\;\; \text{ iff } y = x, \\ \omega'(y) \text{ iff } y \neq x. \end{cases} \tag{3}$$

   iii. If K_1 is in the form of *either* M *or* N, then

$$s_{i+1} = \langle \omega'; q' \cdot p; \mathrm{M}, \mathrm{K}_2, \ldots, \mathrm{K}_m \rangle \text{ or } s_{i+1} = \langle \omega'; q' \cdot (1-p); \mathrm{N}, \mathrm{K}_2, \ldots, \mathrm{K}_m \rangle \tag{4}$$

   and p denotes the probability of choosing the subprogram M.

   iv. If K_1 is in the form of *begin* M; N *end*, then

$$s_{i+1} = \langle \omega'; q'; \mathrm{M}, \mathrm{N}, \mathrm{K}_2, \ldots, \mathrm{K}_m \rangle. \tag{5}$$

   v. If K_1 is in the form of *if* γ *then* M *else* N, then

$$s_{i+1} = \begin{cases} \langle \omega'; q'; M, K_2, \ldots, K_m \rangle \text{ iff } \omega' \models \gamma, \\ \langle \omega'; q'; N, K_2, \ldots, K_m \rangle \text{ otherwise.} \end{cases} \tag{6}$$

   vi. If K_1 is in the form of *while* γ *do* M, then

$$s_{i+1} = \begin{cases} \langle \omega'; q'; \mathrm{M}, \textit{while } \gamma \textit{ do } \mathrm{M}, \mathrm{K}_2, \ldots, \mathrm{K}_m \rangle \text{ iff } \omega' \models \gamma, \\ \langle \omega'; q'; \mathrm{K}_2, \ldots, \mathrm{K}_m \rangle \quad \text{otherwise.} \end{cases} \tag{7}$$

III. If the i-th triple of the sequence is in the form of $s_i = \langle \omega'; q'; \emptyset \rangle$, then s_i is the last element of the sequence. The valuation ω' is the result of the successful computation. The valuation ω' is obtained with the probability q'.

3.4 Algebraic semantics of L_{prob}

Let $\langle \Im, \rho, p \rangle$ be a structure for L_{prob} , K be a program and W be the set of all valuations of variables from the program K. The measure

$$\mu : 2^W \to [0,1] \tag{8}$$

will be called probability distribution on W. Let us consider the set S of all such measures μ on W. Program K is interpreted as a partial function $\mathrm{K}_{\langle \Im, \rho, p \rangle} : S \to S$. If μ is an input distribution, such that $\mu(\omega_i) = \mu_i, i = 1, \ldots, l$, then $\mu' = \mathrm{K}_{\langle \Im, \rho, p \rangle}(\mu)$ is the output distribution such that $\mu'(\omega_i) = \mu', i = 1, \ldots, l$ (Fig. 1).

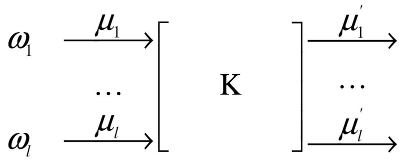


Fig. 1. Probabilistic interpretation of a program

Theorem 1. *[1]. Let $\langle \Im, \rho, p \rangle$ be a structure for L_{prob} with universe $U = \{u_1, u_2, \ldots, u_r\}$. For every program* K$(x_1, \ldots, x_1)$ *interpreted in $\langle \Im, \rho, p \rangle$, we can construct, in an effective way, a matrix $K = [k_{ij}]_{i,j=1,\ldots,l}$, where $l = r^n$, such that for every input probability distribution μ, the output distribution μ' satisfies:*

$$\mu' = \mu \cdot K. \tag{9}$$

An element k_{ij} of matrix K corresponds to the probability that ω_j is the output valuation after computation of program K, provided that the computation starts at the valuation ω_i. The way of construction of the transition matrix K corresponding to program K is described in details in [1,5].
The following lemma enables us to determine, which positions k_{ij} of the matrix K are equal 0.

Lemma 1. *[1]. Let* K *be a program of the form while γ do* M *and let* M^{l+1} $(l = r^n)$ *denote the program*

$$\textit{begin}\ \underbrace{\textit{if}\,\gamma\,\textit{then}\,\mathrm{M};\ldots;\ \textit{if}\,\gamma\,\textit{then}\,\mathrm{M}}_{l+1\ \textit{times}}\ \textit{end.}$$

Then for each distribution μ,

$$[\textit{while}\,\gamma\,\textit{do}\,\mathrm{M}]_{\langle\Im,\rho,p\rangle}(\mu) = [\textit{while}\,\gamma\,\textit{do}\,\mathrm{M}^{l+1}]_{\langle\Im,\rho,p\rangle}(\mu). \tag{10}$$

Furthermore,

$$k_{ij} = 0\ \textit{iff}\ m_{ij}^{l+1} = 0,\ i,j = 1,\ldots,l, \tag{11}$$

where k_{ij} denote elements of the matrix K corresponding to the program K *in the structure $\langle\Im,\rho,p\rangle$, and m_{ij}^{l+1} denote the elements of the matrix* M^{l+1} *corresponding to the program* M^{l+1}.

Moreover, by that lemma we can detect the places in program K, where it loops.

4. Non-deterministic Algorithmic Logic

In this subsection we briefly present a non-deterministic algorithmic language L_{ndtm}, being an extension of L_{alg} $(L \subset L_{alg} \subset L_{ndtm})$. The interpretation of the language L_{ndtm} will be understood as the pair $\langle\Im, U_0\rangle$, where $\Im$ is a structure for first-order language L with a finite universe U and U_0 denotes a subset of U. The non-deterministic interpretations $\langle\Im, U_0\rangle$ will be called *probe* if and only if $U_0 = U$.

4.1 Syntax of L_{ndtm}

We are given a fixed alphabet in which V is a set of individual and propositional variables, Ψ is a set of predicates, and Φ is a set of functors. Let F_o be a set of open formulas and T be set of terms. The set of non-deterministic iterative programs is defined in the same way, as for language L_{alg}, with nondeterministic assignment $x :=?$ and non-deterministic program construction *either* M_1 *or* M_2. The set of all formulas we denote by F_{ndtm}. It contains the classical first-order formulas constructed by means of the connectives $\wedge, \vee, \neg, \Rightarrow$ and quantifiers $\exists, \forall$. It also contains every expression α in the following form:

$$(\exists x)\beta(x), (\forall x)\beta(x), \tag{12}$$

$$(\beta \vee \lambda), (\beta \wedge \lambda), (\beta \Rightarrow \lambda), \neg\beta, \tag{13}$$

$$\Box \mathrm{K}\beta, \Diamond \mathrm{K}\beta, \tag{14}$$

where x is an individual variable, β, λ are arbitrary formulas and K is an arbitrary non-deterministic program.

The informal meaning of the formula $\Box \mathrm{K}\beta$ is: *it is necessary that after realization* K *the formula* β *holds*. The informal meaning of the formula $\Diamond \mathrm{K}\beta$ is: *it is possible that after realization* K *the formula* β *holds*, analogously to [8].
The fact, that the valuation ω in the structure $\langle \Im, U_0 \rangle$ satisfies the formula α will be denoted by $\langle \Im, U_0 \rangle, \omega \models \alpha$ (or simply $\omega \models \alpha$ when no confusion can arise).

The interpretation of open formulas and terms in the language L_{ndtm} is classical and this can be found in [8].
The element $\tau_{\langle \Im, U_0 \rangle}(\omega)$ of U is called *the value of the term* τ *in the structure* $\langle \Im, U_0 \rangle$ *at the valuation* ω . It will be shortly denoted by $\tau(\omega)$ when no confusion can arise.

4.2 Computational semantics of L_{ndtm}

By a *configuration* in structure $\langle \Im, U_0 \rangle$ we shall understand an ordered pair $\langle \omega; \overrightarrow{K} \rangle$, where ω is a valuation, and $\overrightarrow{K} = \{\mathrm{K}_1, \ldots, \mathrm{K}_n\}$ is a finite sequence of programs (which may be empty).

By a *tree of possible computations* of a program $\mathrm{K}(x_1, x_2, \ldots, x_n)$ in the structure $\langle \Im, U_0 \rangle$ with a fixed input valuation ω we mean a tree $Tree(\mathrm{K}, \omega, \Im, U_o)$ (shortly $Tree$), such that the configuration $\langle \omega; \overrightarrow{K} \rangle$ is the root of the $Tree$, *Rest* denotes a sequence of programs (it may be empty) and:

i. If a configuration $\langle \omega'$; *if* γ *then* M *else* N, *Rest*$\rangle$ is a vertex of Tree, then the unique son of this vertex is $\langle \omega'$; M, *Rest*$\rangle$ if $\omega' \models \gamma$ or $\langle \omega'$; N, *Rest*$\rangle$ otherwise.
ii. If a configuration $\langle \omega'$; *begin* M; N *end*, *Rest*$\rangle$ is a vertex of Tree, then the unique son of this vertex is $\langle \omega'$; M, N, *Rest*$\rangle$.
iii. If a configuration $\langle \omega'$; *while* γ *do* M, *Rest*$\rangle$ is a vertex of Tree, then the unique son of this vertex is $\langle \omega'$; M,*while* γ *do* M, *Rest*$\rangle$ if $\omega' \models \alpha$ or $\langle \omega'$;*Rest*$\rangle$ otherwise.
iv. If a configuration $\langle \omega'$;*either* M *or* N,*Rest*$\rangle$ is a vertex of Tree, then the left son of this vertex is $\langle \omega'$; M, *Rest*$\rangle$ and the right son is $\langle \omega'$; N, *Rest*$\rangle$.
v. If a configuration $\langle \omega'; x := \tau, Rest \rangle$ is a vertex of Tree, then the unique son of this vertex is $\langle \omega''; Rest \rangle$, where

$$\omega''(y) = \begin{cases} \tau(\omega') \text{ iff } y = x, \\ \omega'(y) \text{ iff } y \neq x. \end{cases} \tag{15}$$

vi. If a configuration $\langle \omega'; x :=?, Rest \rangle$ is a vertex of Tree, then the sons of this vertex are in the form $\langle \omega''; Rest \rangle$, where

$$\omega''(y) = \begin{cases} u \quad \text{ iff } y = x \wedge u \in U_0, \\ \omega'(y) \text{ iff } y \neq x. \end{cases} \tag{16}$$

vii. If a configuration $\langle \omega' ; \emptyset \rangle$ is a vertex of Tree, then it is a leaf of Tree, i.e. it has no sons.

Every path of the $Tree(\mathrm{K}, \omega, \Im, U_o)$ we called a *computation* of a program K in the structure $\langle \Im, U_0 \rangle$ at the input valuation ω. If $\langle \omega' ; \emptyset \rangle$ is the leaf of the $Tree(\mathrm{K}, \omega, \Im, U_o)$, then the valuation ω' is called the *result* of the corresponding computation.

By the *interpretation* of program K in the structure $\langle \Im, U_0 \rangle$ we shall understand binary relation $\mathrm{K}_{\langle \Im, U_0 \rangle}$ in the set W of all valuations, such that $(\omega, \omega') \in \mathrm{K}_{\langle \Im, U_0 \rangle}$ iff ω' is a result of a computation of program K from the valuation ω in the structure $\langle \Im, U_0 \rangle$. The set of all results of the program K at the valuation ω in the structure $\langle \Im, U_0 \rangle$ we denote by

$$\mathrm{K}_{\langle \Im, U_0 \rangle}(\omega) = \{\omega' : (\omega, \omega') \in \mathrm{K}_{\langle \Im, U_0 \rangle}\} \tag{17}$$

The definition of interpretation of program K for different programs is as follows:

i. If K is of the form $x := \tau$, then

$$\mathrm{K}_{\langle \Im, U_0 \rangle} = \left\{(\omega, \omega') : \omega'(x) = \tau_{\langle \Im, U_0 \rangle}(\omega) \wedge \forall_{y \in V \setminus \{x\}} \omega'(y) = \omega(y)\right\}. \tag{18}$$

ii. If K is of the form $x :=?$, then

$$\mathrm{K}_{\langle \Im, U_0 \rangle} = \left\{(\omega, \omega') : \omega'(x) \in U_0 \wedge \forall_{y \in V \setminus \{x\}} \omega'(y) = \omega(y)\right\}. \tag{19}$$

iii. If K is of the form *while* $\neg\gamma$ *do* $x := x$, then the interpretation of program K will be denoted by I_γ and

$$I_\gamma = \{(\omega, \omega) : \omega \models \gamma\}. \tag{20}$$

iv. If K is of the form *begin* M; N *end*, and $M_{\langle \Im, U_0 \rangle}$, $N_{\langle \Im, U_0 \rangle}$ are the interpretations of the subprograms M, N, then

$$K_{\langle \Im, U_0 \rangle} = M_{\langle \Im, U_0 \rangle} \circ N_{\langle \Im, U_0 \rangle}. \tag{21}$$

v. If K is of the form *if* γ *then* M *else* N and $M_{\langle \Im, U_0 \rangle}, N_{\langle \Im, U_0 \rangle}$ are the interpretations of the subprograms M, N respectively, then

$$K_{\langle \Im, U_0 \rangle} = I_\gamma \cap M_{\langle \Im, U_0 \rangle} \cup I_{\neg\gamma} \cap N_{\langle \Im, U_0 \rangle}. \tag{22}$$

vi. If K is of the form *either* M *or* N , and $M_{\langle \Im, U_0 \rangle}, N_{\langle \Im, U_0 \rangle}$ are the interpretations of the subprograms M, N respectively, then

$$K_{\langle \Im, U_0 \rangle} = M_{\langle \Im, U_0 \rangle} \cup N_{\langle \Im, U_0 \rangle}. \tag{23}$$

vii. If K is of the form *while* γ *do* M, then

$$K_{\langle \Im, U_0 \rangle} = \bigcup_{i=0}^{\infty} \left(I_\gamma \cap M_{\langle \Im, U_0 \rangle} \cup I_{\neg\gamma}\right)^i \circ I_{\neg\gamma}. \tag{24}$$

In the above $\cup$, $\cap$ denote standard operations on sets and $\circ$ denotes the superposition of relations.

For an arbitrary valuation ω in the structure $\langle \Im, U_0 \rangle$ we assume:

$$\langle \Im, U_0 \rangle, \omega \models \Diamond \mathrm{K}\beta \text{ iff } \exists_{\omega' \in K_{\langle \Im, U_0 \rangle}(\omega)} \langle \Im, U_0 \rangle, \omega' \models \beta, \tag{25}$$

$$\langle \Im, U_0 \rangle, \omega \models \Box \mathrm{K}\beta \text{ iff } \forall_{\omega' \in K_{\langle \Im, U_0 \rangle}(\omega)} \langle \Im, U_0 \rangle, \omega' \models \beta \text{ and}$$
$$\text{all computations of the program K at the valuation } \omega \text{ are finite.} \tag{26}$$

Remark 1. The formula $\Box$K**1**, where **1** is any formula of the form $\gamma \vee \neg\gamma$ will be called the *stopping formula* of the program K and it will be denoted by STOP(K). For this formula we have:

$$\langle \Im, U_0 \rangle \models \text{STOP(K) iff all computations of the program K are finite.} \tag{27}$$

5. Results

Example 1. Let the probabilistic distribution on the set $U = \{1, 2, 3\}$ be fixed as:

$$\rho(1) = 1/2, \rho(2) = 1/3, \rho(3) = 1/6.$$

Let us consider an extremely simple program K interpreted in the universe U:

Algorithm 1 Program K

```
while $x \neq 3$ do
    if $x = 1$ then
        $x :=?$
    else
        $x := 1$
    end if
end while
```

It is possible that, the sequence of states of program K looks as follows:

$$\omega(x) = 1, \omega(x) = 2, \omega(x) = 1, \omega(x) = 2, \omega(x) = 1, \omega(x) = 2, ... \tag{28}$$

However, if we consider the program K in the probabilistic structure $\langle \Im, \rho, p \rangle$ we have transition matrix for that program in the form:

$$K = \begin{bmatrix} 0\ 0\ 1 \\ 0\ 0\ 1 \\ 0\ 0\ 1 \end{bmatrix}. \tag{29}$$

The probability that program K halts (when $x = 3$) is exactly 1 and the probability that the program K loops equals 0.

Now, lets consider program K in the non-deterministic structure $\langle \Im, U_0 \rangle$. It is easy to observe that in the tree $Tree(K, \omega, \Im, U_0)$ of possible computations of a program K contains infinite path. It means that, it is possible that program K loops. Thus the formula $\Box$ K$(x = 3)$, expressing fact that the all computations are finite, is not valid in the structure $\langle \Im, U_0 \rangle$.

Now, we can formulate our main results.

Definition 1. *The interpretations $\langle \Im, \rho, p \rangle$ and $\langle \Im, U_0 \rangle$ will be called* consistent *provided that, for each i* $= 1, 2, \ldots, r$ *the following holds:*

$$u \in U_0 \Leftrightarrow \rho(\mu) > 0. \tag{30}$$

Lemma 2. *Assume that the non-deterministic structure $\langle \Im, U_0 \rangle$ is consistent with the probabilistic structure $\langle \Im, \rho, p \rangle$. If*

$$\left\langle \omega^{(0)}; q^{(1)}; \overrightarrow{K}^{(1)} \right\rangle, \left\langle \omega^{(1)}; q^{(2)}; \overrightarrow{K}^{(2)} \right\rangle, \ldots, \left\langle \omega^{(i+1)}; q^{(i+1)}; \overrightarrow{K}^{(i+1)} \right\rangle, \ldots \tag{31}$$

is a probabilistic computation (finite or not) of the program K for the input valuation $\omega^{(0)}$, then

$$\left\langle \omega^{(0)}; \overrightarrow{K}^{(1)} \right\rangle, \left\langle \omega^{(1)}; \overrightarrow{K}^{(2)} \right\rangle, \ldots, \left\langle \omega^{(i+1)}; \overrightarrow{K}^{(i+1)} \right\rangle, \ldots \tag{32}$$

is a non-deterministic computation (finite or not) of the program K for the input valuation $\omega^{(0)}$.

Proof. The proof is by induction on i (with respect to the length of computation) and consists in analysis all possible cases of the program constructions. It is omitted for the sake of this paper compactness.

As an immediate consequence of Lemma 2 we have

Corollary 1. *Let $\langle \Im, \rho, p\rangle$ and $\langle \Im, U_0\rangle$ be probabilistic and non-deterministic interpretation structures for L_{alg} respectively and ω, ω' be valuations. Let μ be a probability distribution such that $\mu(\omega) > 0$. From the above we check at once that in the case, when the output distribution $\mu' = K_{\langle \Im, \rho, p\rangle}(\mu)$ satisfies $\mu'(\omega') > 0$, exists a non-deterministic computation of program K starting with ω and ending at ω'.*

We can also prove the following

Theorem 2. *Let* K *be a program in the algorithmic language L_{alg}. Assume that the non-deterministic structure $\langle \Im, U_0\rangle$ is consistent with the probabilistic structure $\langle \Im, \rho, p\rangle$. For any distribution μ and any valuation ω in the structure $\langle \Im, \rho, p\rangle$, if $\mu(\omega) = 1$ and $\langle \Im, \rho, p\rangle, \omega \models$*K*$(P(\gamma) > 0)$, then $\langle \Im, U_0\rangle, \omega \models \Diamond$*K*$\gamma$.*

Proof. Proof is omitted. It proceeds by induction on the length of the program computation.

Now, we will formulate the non-deterministic analogon of the Lemma 1.

Theorem 3. *Let* K *be a program in the algorithmic language L_{alg}. Assume that the non-deterministic structure $\langle \Im, U_0\rangle$ is consistent with the probabilistic structure $\langle \Im, \rho, p\rangle$. If in the tree of all possible probabilistic computations of the program K, starting with distribution satisfying $\mu(\omega) = 1$, exists a path of length longer than l, where $l = |W| = r^n$, then $\langle \Im, U_0\rangle, \omega \models \neg$* STOP(K).

5.1 Algebraic semantics of L_{ndtm}

In this approach we suggest using the calculus abstract over probabilities. More precisely, we resign from precise probability values by replacing them with terms possibility and necessity. We define formalism by analogy to algebraic semantics of iterative probabilistic algorithms and using in matrices only two values: 0 and e, expressing impossibility and possibility, respectively.

Definition 2. *Let $\langle \{0, e\}, +, \cdot = \rangle$ be the algebraic structure. The operations $\cdot, +$ are defined on the set $\{0, e\}$ in the following way.*

Table 1. Operation 1

$\cdot$	0	e
0	0	0
e	0	e

Table 2. Operation 2

$+$	0	e
0	0	e
e	e	e

The following theorem we will formulate without proof.

Theorem 4. *Let $\langle \Im, U_0\rangle$ be a structure for L_{ndtm} with the universe $U = \{u_1, u_2, \ldots, u_r\}$. Let $\mathrm{K}(x_1, \ldots, x_n)$ be a program interpreted in the structure $\langle \Im, U_0\rangle$ and ω_i, ω_j be any valuations from the set $W = \{\omega_1, \ldots, \omega_l\}, l = r^n$. For every program $\mathrm{K}(x_1, \ldots, x_n)$, we can construct a matrix*

$$E_K = [e_{ij}]_{i,j=1,\ldots,l}, \tag{33}$$

such that $e_{ij} \in \{0, e\}$ and

$$e_{ij} = \begin{cases} e \textit{ iff } (\omega_i, \omega_j) \in K_{\langle \Im, U_0\rangle} \\ 0 \textit{ otherwise.} \end{cases} \tag{34}$$

Proof. The detailed proof is omitted. We only give the main ideas of construction of the transition matrix E_K corresponding to program K:

i. If K is of the form $x := \tau$, then matrix E_K is defined as:

$$e_{ij} = \begin{cases} e \text{ iff } \omega_j(x) = \tau_{\langle \Im, U_0\rangle}(\omega_i) \wedge \forall_{y \in V\setminus\{x\}} \omega_j(y) = \omega_i(y), \\ 0 \text{ otherwise.} \end{cases} \tag{35}$$

ii. If K is of the form $x :=?$, then matrix E_K is defined as:

$$e_{ij} = \begin{cases} e \text{ iff } \omega_j(x) \in U_0 \wedge \forall_{y \in V\setminus\{x\}} \omega_j(y) = \omega_i(y) \\ 0 \text{ otherwise.} \end{cases} \tag{36}$$

iii. If K is of the form *while* $\neg\gamma$ *do* $x := x$, then matrix corresponding to program K will be denoted by I_γ and defined as:

$$e_{ij} = \begin{cases} \text{e iff } i = j \wedge \omega_i \models \gamma, \\ 0 \text{ otherwise.} \end{cases} \tag{37}$$

iv. If K is of the form *begin* M; N *end*, and the matrices E_M and E_N for the subprograms M, N are constructed, then

$$E_K = E_M \cdot E_N. \tag{38}$$

v. If K is of the form *if* γ *then* M *else* N, and the matrices E_M and E_N for the subprograms M, N are constructed, then

$$E_K = I_\gamma \cdot E_M + I_{\neg\gamma} \cdot E_N. \tag{39}$$

vi. If K is of the form *either* M *or* N, and the matrices E_M and E_N for the subprograms M, N are constructed, then

$$E_K = E_M + E_N. \tag{40}$$

vii. If K is of the form *while* γ *do* M, then the matrix E_K is defined as:

$$E_K = \lim_{i\to\infty} \sum_{j=0}^{i} \left(I_\gamma \cdot E_M + I_{\neg\gamma}\right)^i \cdot I_{\neg\gamma}. \tag{41}$$

The reader can easily verify that the limit in the equation 41 always exists.

Example 2. Let us reconsider program K from Example 1. The transition matrix E_K corresponding to the program K, determined in accordance with the above rules, is as follows:

$$E_K = \begin{bmatrix} 0 & 0 & e \\ 0 & 0 & e \\ 0 & 0 & e \end{bmatrix}. \tag{42}$$

6. Final remarks

The formalism proposed in subsection 5.1 is useful in situations in which we interest if it is possible that program K halts and we do not have to know the accurate values of probabilities, except the fact that they are positive (cf. [1]).

Theorem 2 and Theorem 3 point at a direction for further work. The main goal of our research is to explain the dependencies between non-deterministic and probabilistic interpretation of the iterative programs. It will be the subject of next paper.

References

[1] Dańko W.: The Set of Probabilistic Algorithmic Formulas Valid in a Finite Structure is Decidable with Respect to Its Diagram, Fundamenta Informaticae 19, 1993, pp. 417–431.

[2] Feldman Y. A., Harel D.: A Probabilistic Dynamic Logic, Journal of Computer and System Sciences 28, 1984, pp. 193–215.

[3] Hare D., Pratt V. R.: Nondeterminism in Logics of Programs, Proc. of the 5th Symp. on Principles of Programming Languages SIGACT - SIGPLAN, ACM, 1978, pp. 23–25.
[4] Harel D., Kozen D., Tiuryn J.: Dynamic Logic, MIT Press, MA, 2000.
[5] Koszelew J.: The Methods for Verification Properties of Probabilistic Programs, Ph.D. thesis, Inst. of Comput. Sci., Polish Academy of Sciences, 2000.
[6] Kozen D.: Semantics of probabilistic programs, Journal of Computer and System Sciences 22, 1981, pp. 328–350.
[7] Kozen D.: A Probabilistic PDL, Journal of Computer and System Sciences 30(2), 1985, pp. 162–178.
[8] Mirkowska G., Salwicki A.: Algorithmic Logic, D. Reidel Publ. Co. & PWN, 1987.

PROBABILISTYCZNE I NIEDETERMINISTYCZNE SEMANTYKI DLA PROGRAMÓW ITERACYJNYCH

Streszczenie W pracy rozważane są, na gruncie logiki programów, probabilistyczne i niedeterministyczne interpretacje programów iteracyjnych. Uwaga autorów skupia się na związkach miedzy tymi interpretacjami. Punktem wyjścia są formalne definicje semantyk dla obu podejść. Główny nacisk został położony na wyrażalność własności stopu w tych semantykach.

Słowa kluczowe: Słowa kluczowe: program niedeterministyczny, program probabilistyczny, obliczenie probabilistyczne, obliczenie niedeterministyczne

Artykuł zrealizowano w ramach pracy badawczej S/WI/3/2008.

ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2009
Informatyka – Zeszyt 4

Andrzej Sawicki[1], Piotr Zubrycki[1], Alexandr Petrovsky[1]

DESIGN OF TEXT TO SPEECH SYNTHESIS SYSTEM BASED ON THE HARMONIC AND NOISE MODEL

Abstract: This is a proposal of concatenative text to speech synthesizer for the Polish language, based on diphones and "Harmonics and Noise Model"(HNM). HNM has been successfully applied on a speech encoder and decoder, resulting in a high-quality of processed speech at low bit rate. Applying this model to speech synthesis system allows obtaining good quality of synthesized speech, and the small size of database parameters.
The proposed project consists of two main modules. The Natural Language Processing (NLP) is used to analyse and convert the written text for phonemes and diphones using morphological rules. NLP discovers at the same time prosodic features for later modification of synthesized speech parameters in order to obtain the stress and voice intonation. The second section is a synthesis system, derived from speech decoder, preceded by a system of adapting the parameters of speech based on prosodic rules.
The system of speech synthesis from the parameters is working in the frequency domain and uses the frequency spectrum envelope, which easily allows modifying the frequency, amplitude and duration of the signal when applying the prosodic rules. The algorithm of continuous phase designation at the speech frame borders allows concatenating portions of synthesized speech and diphones without phase distortion on the merger. Speech synthesizer operates on the diphone database, created applying fragmentation of recorded speech signal representing the pairs of phonemes. Sounds related to diphones are analyzed by speech encoder. It provides the parameters that described harmonic and noise components of speech, using the linear prediction filter LSF coefficients, resulting in a small size of diphone database.

Keywords: Speech synthesis, TTS, harmonic and noise model

1. Introduction

Most of modern Text To Speech (TTS) systems are based on unit concatenation[1]. Concatenative text-to-speech systems are designed to produce speech by concatenating small segmental units of speech, such as phonemes, diphones or triphones. TSS systems uses database of recorded, segmented and labeled utterances and words. The choice of unit size motivated by vocabulary is a key element in TTS systems for

[1] Faculty of Computer Science, Bialystok Technical University, Biaystok

improving the synthesis quality and meeting storage requirements. Good quality of produced speech, close to the original voice, assured system with huge databases of recorded speech and large concatenation speech units.

Such model however, has large memory and computing requirements, and its implementation especially in mobile systems is problematic. Small units, like diphones, gives smaller database and computational requirements, but cause several problems, such as distortions at the concatenation points. Distortions can be reduced through select suitable speech model, which provides spectral smoothing between combined parts of speech.

In this paper we propose a concatenative Text To Speech synthesiser for the Polish language, based on diphones and Harmonics and Noise Model (HNM)[8] of speech.

HNM has been successfully applied on a speech encoders and decoders, resulting in a high-quality of processed speech at low bit rate [2]. Applying this model to TTS synthesis system as essential of signal processing module, allows to get good quality of synthesised speech, and the small size of database parameters. The size of the speech database may be reduced through record only HNM speech coefficients: harmonic and noise envelopes, and pitch frequencies of diphones.

The proposed project of TTS system consists of two main modules. The Natural Language Processing module (NLP) is used to text analyse and to provide to the synthesiser necessary prosodic and phonemic information. HNM Synthesis module is used to produce synthetic speech.

The paper is organised as follows. Section 2 of document provides description of Natural Language Processing module in TTS system for Polish language. Section 3 describes in detail HNM for speech analysis and synthesis. In section 4 process of diphones database creation is introduced. In section 5 TTS synthesis system is presented. Section 6 comprises conclusions of the article.

2. Natural Language Processing

Natural Language Processing (NLP) module is responsible for text analysing and its conversion to a phonetic transcription. First the incoming text must be accurately converted to its phonemic and stress level representations. Written text and all symbols, numbers, abbreviations and non-text expressions should be converted into speakable forms. This includes determination of word boundaries, syllabic boundaries, syllabic accents, and phonemic boundaries. Then prosody properties of text should be discovered for proper intonation and stress in synthesised speech pronunciation.There are numerous methods that have been proposed and implemented for the text processing for English, especially. For Polish language, see [5].

2.1 Text preprocessing and normalisation

Text normalization encompasses all aspects of conversion from the mixture of abbreviations, symbols, numbers, dates, and other no orthographic entities of text into a appropriate orthographic transcription. Text is divided into phrases, using end of phrase punctuation marks as '.', ',', '?', '!'. Then sentences are splitted into individual tokens based on whitespaces and punctuation marks. Each token can be classified into one of the following group:

– Words
– Symbols, E.g.: =, +, -, %, $.
– Abbreviations, E.g.: mgr., n.p.
– Numbers, E.g.: 1,2,3 etc.
– Time, E.g.: 12:34
– Date, E.g.: 01/02/2008, 12.03.06, 2008 r.,
– URLs and E-mails, E.g.: somebody@domain.com

Identified tokens we have to expand to full text, using lexicon and rules, e.g. + - plus, 2 - dwa (eng. two), 12:34 - dwunasta trzydzieści cztery (eng. thirty four past twelve) , 01/02/2008 - pierwszy luty dwa tysiące osiem (eng. the first of February two thousand eight).

2.2 Grapheme to phoneme conversion

Pronunciation of a words may be determined either by a lexicon (a large list of words and their pronunciations) or by letter to sound rules. In our implementation for Polish language, grapheme to phoneme rules are the basis of the conversion process. For nonstandard and foreign language words, lexicon with direct phoneme translation is used.

In this article we use SAMPA (Speech Assessment Methods Phonetic Alphabet) symbols of phonemes, which mapping symbols of the International Phonetic Alphabet onto ASCII codes. The Polish language acoustic system comprises 37 phonemes, included eight vowels:

– Oral: a, o, u, e, I, i, pronounced [6] as in words *para, bok, paru, era, dary, lis* .
– Nasal: e , o , pronounced as in words *idę, tą*.

Consonants comprise 29 sounds:

- Plosives: p, b, t, d, k, g, pronounced as in words *pani, bak, tama, dam, kura, noga*.
- Fricatives: v, f, s, z, Z, S, z', s', x, pronounced as in words *wam, fajka, sam, koza, żona, szary, kozia, Kasia, chór*.
- Affricates: ts, tS, ts', dz, dZ, dz', pronounced as in words *koca, lecz, lać, rydza, dżem, działa*.
- Nasals: m, n, n', N, pronounced as in words *mam, len, bańka, bank*.
- Approximants: w, l, r, j, pronounced as in words *łam, pal, rak, daj*.

Our TTS system use diphones for concatenate speech synthesis. Diphones are adjacent pair of phonemes, selected from the stationary area of first phone to stationary area of second phone in recording.

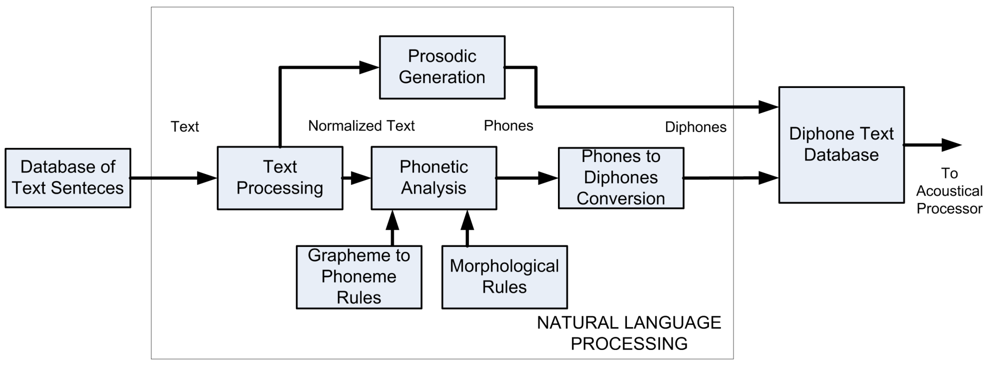


Fig. 1. Natural language processing module

Processing of natural language (NLP) is based on the model proposed by A.W. Black and P. Taylor in the Festival [4] TTS system and used in open source software speech synthesizer eSpeak [14]. Text analysis and prosody annotation for Polish language is based on the work of S. Grocholewski and G. Demenko [5]. Our system use for tests and research database of recorded texts in Polish language 'CORPORA' developed by prof. S. Grocholewski [3]. Diagram of proposed NLP module is presented in fig.1.

3. Harmonic and noise model for speech analysis and synthesis

3.1 Speech analysis

High quality speech coding at low bit-rates is major interest of speech coding methods. Speech signal modelling based on HNM was presented in [8]. Speech is divided into two subbands by the maximum voiced frequency, lower band is considered fully voiced and upper band fully unvoiced. From the speech production point of view it is clear, that both voiced and unvoiced components are present in whole speech band. This idea was used by Yegnanarayana et. all [9] in speech decomposition method into voiced and noise components. Decomposition of speech is performed on excitation signal approximated with use of inverse linear prediction filter. Idea of work is to use an iterative algorithm based on Discrete Fourier Transform (DFT)/Inverse Discrete Fourier Transform (IDFT) pairs for noise component estimation. Voiced component of excitation is obtained by subtracting noise component from LP residual. These approaches were designed without taking into account time-varying fundamental frequency and harmonic amplitudes.

In this paper we present another approach to speech decomposition into voiced and noise components and its application to speech synthesis system. As the methods presented in [9],[10] our method considers voiced and noise components present in whole speech band. Pitch-Tracking modification is applied to standard DFT in order to provide spectral analysis in harmonic domain rather than frequency domain. Voiced component parameters (i.e. harmonics amplitudes and phases) are estimated in harmonic domain. Estimation is done every 16ms. Amplitudes and phases of harmonics are interpolated between points of estimation. Voiced component is generated with time-varying frequency and harmonics amplitudes. After voiced component generation decomposition of speech signal is done in time domain. An iterative algorithm is used for decomposition in order to obtain exact components separation. Time-domain speech components separation and voiced component modelling method is sensitive to pitch estimation errors, thus precise and accurate pitch detection algorithm is needed. A robust pitch detection method based on tuning pitch frequency to its harmonics presented in [11],[2] is used.

Pitch estimation and tracking. Pitch tracking algorithm operates both in time and frequency domain. Preliminary pitch estimation is taken every 8ms using autocorrelation method. This estimate is used to refine pitch frequency using algorithm working in spectral domain similar to the one proposed in [12]. In order to prevent Gross Pitch Errors (GPE) and pitch track distortions a tracking algorithm is used. Scheme of pitch estimation and tracking algorithm is shown on fig. 2. First, the autocorrelation vector is computed. In order to improve robustness of the algorithm low-pass filtering and signal clipping operation are performed according to Sondhi [13]. Autocorrelation vector is computed in interval corresponding to possible fundamental frequency values (typical from 60 to 500Hz) using formula:

$$R(k) = \sum_{n=0}^{N-1} s(n)s(n+k), k = -l..l, \tag{1}$$

where l is maximum lag corresponding to minimal fundamental frequency.

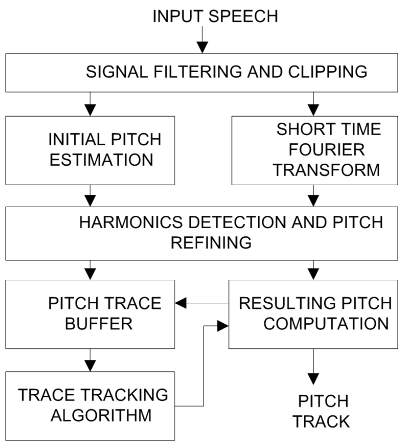

INPUT SPEECH
SIGNAL FILTERING AND CLIPPING
INITIAL PITCH ESTIMATION
SHORT TIME FOURIER TRANSFORM
HARMONICS DETECTION AND PITCH REFINING
PITCH TRACE BUFFER
RESULTING PITCH COMPUTATION
TRACE TRACKING ALGORITHM
PITCH TRACK

Fig. 2. Pitch estimation algorithm

Length of autocorrelation window is 32ms and 16ms before and after the frame is needed, thus algorithm operates with delay of 16ms. This approach to autocorrelation computation enables transient frame detection, because resulting vector is not symmetric. If the speech segment is transient maximum of autocorrelation is available only at one side of vector, and the side depend on the frame is beginning or ending of voiced segment. Initial pitch estimation is computed as weighted mean of maximums of autocorrelation sequence on left and right side.

After initial estimation input speech signal is weighted in 256-point time window and STFT is computed. Initial pitch value is tuned to all present pitch harmonics [2]. In case of inability to identify at least two out of four leading pitch frequency harmonics, the segment is considered unvoiced. Refined pitch value F_r for each weighting window is identified with the harmonic factor, which can be understood as adequacy of the estimation:

$$h_f = \frac{n_h}{n_{max}} \tag{2}$$

where n_h is number of present harmonics, n_{max} is number of all possible harmonics with given pitch.

False pitch frequency estimations got during speech flow pauses are discarded on the base of analysis of the weighting factors of pitch frequency estimations and analysis of the values of the input signal level, of the speech and silence levels. In order to prevent gross errors and provide better quality, pitch estimation is performed with a delay of two analysis windows. Estimations of the pitch frequency are included in the current track in case the difference between neighbouring windows does not exceed the allowed one. Trace tracking estimation of pitch frequency is calculated using linear approximation of current trace according to the least-square method. The condition determining end of the trace tracking is checked by availability of preliminary estimations to the right of the window being analysed and by harmonic factors. Resulting pitch frequency is determined as:

$$F_0 = h_f F_r + (1 - h_f) F_t \tag{3}$$

where F_r is refined pitch; F_t is trace tracking estimation.

Pitch-tracking modified DFT. Pitch-Tracking modified DFT transform providing analysis in spectral domain is given by:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi nkF_0}{F_s}}, k = 0..K \tag{4}$$

where $X(k)$ is k-th spectral component corresponding to k-th harmonic, $x(n)$ is input signal, N is transformation length, F_s is sampling frequency, F_0 is fundamental frequency. Kernel of transformation has to be modified in case tracking analysis. Argument of exponential can be written as follows:

$$\varphi(n,k) = \sum_{i=1}^{n} \frac{2\pi k(F_0(i) - F_0(i-1))}{2F_s}, n \neq 0 \tag{5}$$

$\varphi(n,k) = 0, \texttt{for:} n = 0$, and $F0(i)$ is fundamental frequency at the time specified by i. Transformation providing tracking harmonic analysis is given as follows:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\varphi(n,k)}, k = 0..K \tag{6}$$

Non-orthogonal transformation kernel can cause energy leakage to neighbouring spectral lines. Time-Varying window is used in order to deal with leakage phenomenon. Idea of this solution is to design a spectral window, which follows fundamental frequency changes. Window length is chosen to contain at least two pitch periods. Experiments showed, that window length should be approximately 2.5 pitch period which is a trade-off between spectral resolution and computational efficiency. Good results could be achieved when Kaiser window is used as a prototype [11]:

$$w(n) = \frac{I_0(\beta\sqrt{1-(\frac{2x(n)-(N-1)}{(N-1)})^2})}{I_0(\beta)}, n = 0..N-1 \qquad (7)$$

where N is window length, β is window parameter, $I_0(x)$ is zero order Bessel function, $x(n)$ is a function enabling time-varying feature, given as:

$$x(n) = \frac{\varphi(n-1)}{2\pi\bar{F}_0\frac{N}{F_s}} \qquad (8)$$

where $\phi(n,1)$ is computed using formula (5), $\bar{F}_0$ is average fundamental frequency in analysis frame.

Decomposition algorithm. In this paper we present solution which is based on continuous harmonic component generation. Continuous generation of voiced component is performed with a delay of 16ms which is necessary to iterative algorithm. Proposed method performs decomposition in whole speech band, which leads to more accurate representation of speech. Synthesised signal sounds more natural. Speech signal decomposition scheme is shown on figure 3.

First step of decomposition is pitch tracking. This information is passed to iterative decomposition algorithm. It performs decomposition every 16ms. First step of decomposition is speech windowing with time-varying window. Centre of time window is set every 16ms. In order to reduce leakage length of window is chosen as integer multiple of pitch periods. Pitch-Tracking Modified DFT every 16ms gives an information about instantaneous amplitudes and initial phases of the harmonics. For synthesis of the harmonic component a set of time-varying oscillators can be used:

$$h(n) = \sum_{k=0}^{K} A_k(n)cos(\varphi(n,k) + \Phi_k) \qquad (9)$$

where phase $\varphi(n,k)$ is determined using formula (5). While pitch harmonics amplitudes estimation is performed every 16ms instantaneous amplitudes of harmonics have

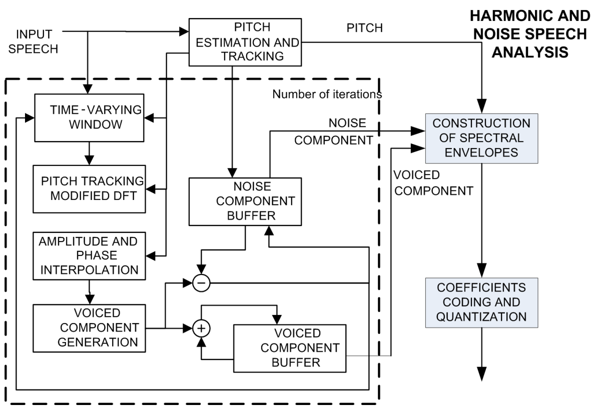

INPUT SPEECH
PITCH ESTIMATION AND TRACKING
PITCH
HARMONIC AND NOISE SPEECH ANALYSIS
Number of iterations
TIME-VARYING WINDOW
NOISE COMPONENT
CONSTRUCTION OF SPECTRAL ENVELOPES
VOICED COMPONENT
PITCH TRACKING MODIFIED DFT
NOISE COMPONENT BUFFER
AMPLITUDE AND PHASE INTERPOLATION
COEFFICIENTS CODING AND QUANTIZATION
VOICED COMPONENT GENERATION
VOICED COMPONENT BUFFER

Fig. 3. Harmonic and noise speech analysis.

to be computed using interpolation algorithm. Piecewise Cubic Hermite Interpolation method is used, as it can be easily implemented in real-time applications. Having harmonic component, noise component is defined as a difference between input speech signal and harmonic component:

$$r(n) = s(n) - h(n) \tag{10}$$

Voiced component is stored in voiced component buffer and noise component is stored in noise component buffer. After first iteration noise component still consists of some voiced component. This is mainly due inability of perfect reconstruction of harmonic amplitudes tracks imposed by amplitudes variations. In the next iterations noise component is processed in the same way as original speech in first iteration. Remaining in noise voiced component is subtracted from noise component. Output voiced component is sum of harmonic components from all iterations and noise component is residual from last iteration.

3.2 Speech synthesis

The periodic spectrum is created by a set of sinusoidal generators working at variable frequency changing linearly within time window, using equation:

$$H(n) = \sum_{k=0}^{K} A_k(n) cos(\varphi(n,k) + \Phi_l) \tag{11}$$

Correctly calculated phases of pitch harmonics avoid any reverberation in the synthesised speech, especially in diphone concatenation points. In proposed model

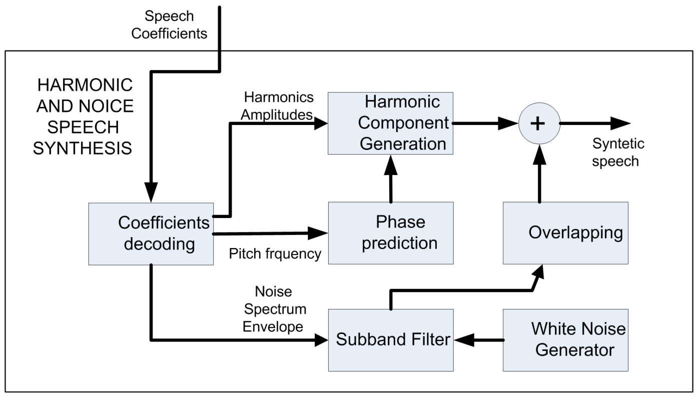


Fig. 4. Harmonic and noise speech synthesis

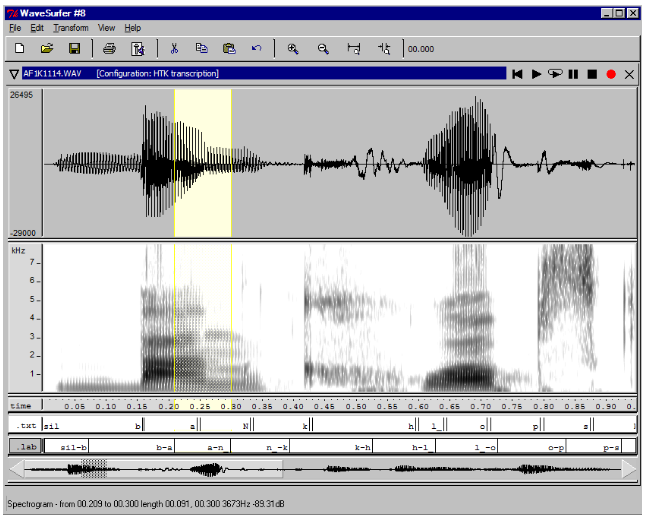
phases are modelled by a parabolic temporal model of instantaneous phase witch take changes on pitch from frame to frame Phase of k+1'th frame is predicted from k'th frame. New phase is calculated on time-varying contribution of the frequency trajectory:

$$\Phi_l(t) = \frac{\omega_l^{k+1} - \omega_l^k}{2T} t^2 + \omega_l^k t + \phi_l^k \tag{12}$$

where T is frame length, and ϕ_l^k is phase offset at the begin of k'th frame. An initial phases for voiced groups of speech frames are chosen randomly. The unvoiced speech is synthesised by the bandpass filtering of white noise. The voiced and the unvoiced components are added. Detailed view of presented speech synthesiser can be found in [2].

4. Diphone Database Creation.

For diphone database creation we used database of Polish Speech CORPORA [3] , projected by prof. Grocholewski. Database consists 45x365 segmented and labelled utterances. Files in database were recorded using 12 bits resolution and frequency 16 kHz. In Polish language there are 37 phonemes. We use additional symbol sil (#) in order to indicate silence at beginning and end of the words, thus the optional number of phoneme to phoneme connections is about 1400. In [3] it is suggested, that not all combination of neighbouring phonemes occur in polish language, or such diphones happen very rarely. Finally, our diphone database consist of 1270 diphones.


WaveSurfer #8
File Edit Transform View Help
00.000
AF1K1114.WAV [Configuration: HTK transcription]
26495
-29000
kHz
time 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70 0.75 0.80 0.85 0.90
.txt sil b a N k h l_ o p s
.lab sil-b b-a a-n_ n_-k k-h h-l_ l_-o o-p p-s

Spectrogram - from 00.209 to 00.300 length 00.091, 00.300 3673Hz -89.31dB

Fig. 5. Waveform diphone labelling using WaveSurfer

Text of database sentences was converted to diphones using NLP module. Preliminary waveform marker description (e.g. beginnings and endings of the particular phoneme) is given in the Corpora database. Proposed system uses diphones as a basic units, thus accurate determination of the exact boundary locations of a diphone in waveform should be performed manually, as shown in figure 5. We use open source program WaveSurfer [7] for waveform labelling. Based on labelled waveform, diphone database for speech synthesis is prepared, using harmonic and noise speech analysis module.

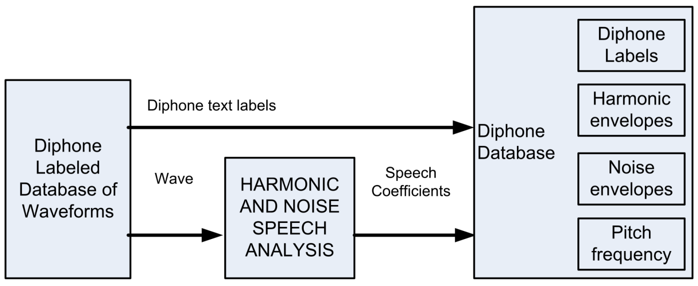


Fig. 6. Diphone database creation.

Each diphone in database is described by set of 16 milliseconds frames included coefficients of speech model:

1. harmonic envelope
2. harmonic gain
3. noise envelope
4. noise envelope gain
5. pitch frequency

For each diphone additional information about its duration is stored in the database. The duration is determined by waveform labels.

5. Text To Speech synthesis using Harmonic and Noise Model.

TTS system consist of 3 main components described above: NLP section, HNM synthesis module and diphone database. The diphone database is created for one speaker. All utterances are analysed using method described in section 3.1 and the database is created as described in section 4. As a result of text processing in NLP module, we get diphone labels and prosody description. The prosody description gives information about stress of the syllable and necessary modifications of the parameters i.e. relative changes of a F_0, duration and gain for particular diphone. Using diphone text labels, a set of frames for actual diphone is selected from database. Speech model coefficients for each frame are transformed in accordance with prosody descriptions. It is worth notice, that using harmonic model makes diphone transformation process easy. Gain and pitch modification is simply straightforward, modification of the diphone duration is done by changing the time offset between frame coefficients (i.e. single frame time-scaling). Converted coefficients are passed to the signal processing module, where synthetic speech is generated in accordance with the HNM, as was described in section 3.2.

For example for given text:

”Artykuł przedstawia projekt konkatenacyjnego syntezatora mowy z tekstu dla języka polskiego.”

phonetical description, using presented above phoneme notation, with word stress (appointed as: ‘) is following:

” art‘Ikuw pSetst‘avja pr‘ojekt koNkatenatsIjn‘ego sIntezat‘ora m‘ovI s t’ekstu dla je z‘yka polski‘ego”.

And diphone transcription is following: *”#-a a-r r-t t-I I-k k-u u-w w-# #-p p-S S-e e-t t-s s-t t-a a-v v-j j-a a-# #-p p-r r-o o-j j-e e-k k-t t-# #-k k-o o-N N-k k-a a-t t-e e-n n-a a-ts ts-I I-j j-n n-e e-g g-o o-# #-s s-I I-n n-t t-e e-z z-a a-t t-o o-r r-a a# #-m*

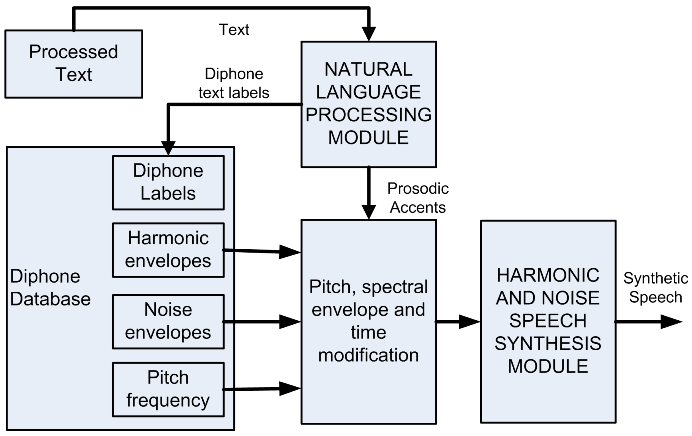


Fig. 7. TTS Synthesis

m-o o-v v-I I-# #-s s-# #-t t-e e-k k-s s-t t-u u-# #-d d-l l-a a-# #-j j-e e -z z-I I-k k-a a# #-p p-o o-l l-s s-k k-i i-e e-g g-o o-#”

Such list of diphones is synthesised using HNM model described in section 3.

6. Conclusions

In this article we propose new approach to the topic of concatenative text to speech synthesiser for the Polish language. System based on Harmonics and Noise Model of speech allow to accurate represent speech signal for diphones database building. Signal processing in frequency domain and instantaneous interpolation of harmonic amplitudes and phases allow to smooth concatenation of diphones. The HNM-based speech analysis/synthesis system proved its ability to produce high quality of synthetic speech [12], almost indistinguishable from the original one. The performance of the proposed TTS system mostly depends on NLP module. Time and frequency modification of diphones can be done separately and without any loose of quality. Presented approach can be successfully used in high-quality speech synthesis applications for TTS system.

Our TTS system is still in development. Especially prosody parsing and diphones modification in accordance with prosody annotations have need of improvement. Current version of NLP module gives information about stress in syllables, phrase intonation description function is under development. While voiced component modification is easy task, unvoiced component in current version of the system can change only its duration which can cause artifacts. Future works include improvement of voiced component modification method. Applying effective methods of speech compression using vector quantization for database size reduction is also under investigation.

Small database size and effective algorithms of speech generation allow to implement presented system in embedded devices.

References

[1] Dutoit T., An Introduction to Text-to-Speech Synthesis, Kluwer Academic Publishers, 1997.

[2] Petrowsky A., Zubrycki P., Sawicki A.: Tonal and Noise Components Separation Based on a Pitch Synchronous DFT Analyzer as a Speech Coding Method, Proceedings of ECCTD, 2003, Vol. III, pp. 169-172.

[3] Grocholewski S.: Założenia akustycznej bazy danych dla języka polskiego na nośniku CD-ROM, Mat. I KK: Głosowa komunikacja człowiek-komputer, Wrocław 1995, s. 177-180

[4] A. Black and P. Taylor: Festival Speech Synthesis System: system documentation (1.1.1), Human Communication Research Centre Technical Report HCRC/TR-83, 1997.

[5] Demenko, G. Grocholewski, S. Wagner, A. Szymanski M.: Prosody annotation for corpus based speech synthesis. [in:] Proceedings of the Eleventh Australasian International Conference on Speech Science and Technology, New Zealand. Auckland, 2006.

[6] M. Wiśniewski: Zarys fonetyki i fonologii współczesnego języka polskiego, wyd. Uniwersytetu Mikołaja Kopernika, Toruń, 2007.

[7] Sjolander, Kyre / Beskow, Jonas: Wavesurfer - an open source speech tool, In ICSLP-2000, vol.4, 464-467

[8] Y. Stylianou, Applying the Harmonic Plus Noise Mode in Concatenative Speech Synthesis, IEEE Trans. on Speech and Audio Processing, vol. 9, no 1., 2001.

[9] B. Yegnanarayana, C. d'Alessandro, V. Darsions An Iterative Algorithm for Decomposiiton of Speech Signals into Voiced and Noise Components, IEEE Trans. on Speech and Audio Coding, vol. 6, no. 1, pp. 1-11, 1998.

[10] P.J.B. Jackson, C.H. Shadle, Pitch-Scaled Estimation of Simultaneous Voiced and Turbulence-Noise Components in Speech, IEEE Trans. on Speech and Audio Processing, vol. 9, no. 7, pp. 713-726, Oct. 2001

[11] V. Sercov, A. Petrovsky, An Improved Speech Model with Allowance for Time-Varying Pitch Harmonic Amplitudes and Frequencies in Low Bit-Rate MBE Coders, in Proc. of the 6ht European ?onf. on Speech Communication and Technology EUROSPEECH'99, Budapest, Hungary, 1999, pp. 1479-1482.

[12] P. Zubrycki, A. Petrovsky Analysis/synthesis speech model based on the pitch-tracking periodic-aperiodic decomposition, in Information processing and security systems (Khalid Saeed, Jerzy Peja eds.) Springer Verlag, Heidelberg 2005, pp. 33-42

[13] M.M. Sondhi, New Methods of Pitch Extraction, IEEE Trans. on Audio and Electroacoustics, vol. AU-16, no. 2, pp. 262-266, 1968.

[14] Espeak, eSpeak text to speech, http://espeak.sourceforge.net/[viewed 15/09/2009]

KONCEPCJA UKŁADU SYNTEZY MOWY Z TEKSTU OPARTEGO NA MODELU HARMONICZNE I SZUM

Streszczenie: Artykuł przedstawia projekt konkatenacyjnego syntezatora mowy z tekstu dla języka polskiego, opartego na difonach i modelu Harmoniczne i Szum. Model Harmoniczne i Szum został z powodzeniem zastosowany w układzie kodera i dekodera mowy, dając w rezultacie dobrą jakość przetwarzanej mowy przy niskiej przepływności bitowej. Zastosowanie tego modelu do układu syntezy mowy pozwala na uzyskanie dobrej jakości syntezowanej mowy, oraz niewielki rozmiar bazy parametrów.
Układ składa się z dwch głównych modułów. Moduł Naturalnego Przetwarzania Języka służy do analizy i zamiany tekstu pisanego na fonemy oraz difony, przy wykorzystaniu reguł morfologicznych. Procesor tekstu wyznacza jednocześnie warunki prozodii związane z późniejszą modyfikacją parametrów syntezowanego głosu w celu uzyskania akcentowania i intonacji. Drugim układem jest moduł syntezy, oparty na dekoderze mowy poprzedzonym systemem adaptacji parametrów mowy w oparciu o wyznaczone wcześniej reguły prozodyczne.
Układ syntezy mowy z parametrw działa w dziedzinie czstotliwości i bazuje na obwiedni spektrum, co w prosty sposób pozwala na modyfikację czstotliwości, amplitudy i czasu trwania sygnału przy stosowaniu reguł prozodycznych. Algorytm wyznaczania ciągłej fazy na granicach ramek sygnału mowy pozwala na łączenie fragmentów syntezowanej mowy oraz poszczególnych difonów bez zniekształceń fazowych na połączeniu.
Syntezator mowy operuje na bazie difonów, stworzonej na podstawie fragmentaryzacji nagranego sygnału mowy na części, reprezentujące połączenia par fonemów. Dźwięki odpowiadające difonom są analizowane przez moduł analizy mowy. Dostarcza on ciąg parametrów reprezentujących harmoniczne i szumowe komponenty sygnału mowy, opisane za pomocą filtrów liniowej predykcji i współczynników LSF, dając w rezultacie niewielkiej wielkości bazę difonów.

Słowa kluczowe: Synteza mowy, model harmoniczne i szum

Artykuł zrealizowano w ramach pracy badawczej W/WI/6/09.

ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2009
Informatyka – Zeszyt 4

Marcin Skoczylas[1,2], Roberto Cherubini[1], Silvia Gerardi[1]

AUTOMATIC DETECTION OF UNSTAINED VIABLE CELLS IN PHASE-CONTRAST MICROSCOPE IMAGES

Abstract: Irradiation of cultured mammalian cells one-by-one with a known number of ions, down to one-ion per single-cell, is a useful experimental approach to investigate the low-dose ionizing radiation exposure effects and to contribute to a more realistic human cancer risk assessment. Mammalian cells (specifically, Chinese hamster V79 cells) are seeded and grown as a monolayer on a mylar surface used as bottom of the special designed holder, having as cover an other mylar foil and allowing the cell culture to be in wet and sterile conditions. Manual recognition of unstained cells in bright-field is a time consuming procedure, therefore a parallel algorithm has been conceived and developed in order to speed-up this step of the irradiation protocol and increase the number of cells that can be irradiated during an accelerator run. Many technical problems have been faced to overcome the complexity of the images to be analyzed. The unstained cells have to be discriminated in an inhomogeneous background, among many disturbing bodies mainly due to the mylar surface roughness and culture medium. Additionally, cells can have various shapes depending on how they attach on the surface, which phase of the cell cycle they are in and on cell density, thus making the detection task more difficult.

Keywords: unstained cell recognition, charged-particle microbeam, image analysis

1. Introduction

It is known that common people are continuously exposed to various sources of ionizing radiation which may be classified in two major groups: natural (background) and man-made radiation sources [5]. Natural sources of radiation, including terrestrial (above all radon gas), internal radiation sources and cosmic rays, constitute the major contribute of exposure of the population.

At the low doses (and low dose rates) relevant to environmental and occupational radiation exposure (0–50 mSv), which are of practical concern for radiation protection, very few cells in the human organism experience more than one traversal by densely ionising particles in their lifetime [3].

[1] Laboratorio di Radiobiologia, INFN-Laboratori Nazionali di Legnaro, Legnaro-Padova, Italy
[2] Faculty of Computer Science, The Białystok Technical University, Białystok, Poland

Up to now no experimental and epidemiological data about low doses are available and so human cancer risk evaluation at low-doses derives from extrapolation of experimental and epidemiological data collected at high doses by means of *in-vitro* and *in-vivo* conventional cell irradiations as well as from studies on Uranium miners, Japanese atomic bomb survivors, nuclear fallout accidents. In practice, in radiation protection a linear no-threshold (LNT) risk model is assumed in the low dose region in a precautionary way.

Recent *in-vitro* radiobiological results following direct investigations of low dose effects seem to indicate a non-linear response to low dose exposure as a consequence of phenomena acting in this dose region: hypersensitivity, induced radioresistance, adaptive response and bystander effect.

Irradiation of cultured mammalian cells one-by-one with a known number of ions, down to one-ion per single-cell, is a useful experimental approach to investigate the low-dose ionizing radiation exposure effects and to contribute to a more realistic human cancer risk assessment. In the past 15 years, the development and use of accelerator-based charged-particle microbeams as a tool for low-dose studies has generated much interest in the scientific community [3]. In this framework, a horizontal single-ion microbeam facility for single-cell irradiations has been designed and installed at the Istituto Nazionale di Fisica Nucleare, Laboratori Nazionali di Legnaro (INFN-LNL), 7MV Van de Graaff CN accelerator [4].

Cell recognition is a fundamental task to be accomplished to perform single-ion microbeam cell irradiation. As a distinctive feature of the INFN-LNL facility, cell recognition is performed without using fluorescent staining and UV light. In particular, this is based on an inverted phase-contrast optical microscope coupled with a grayscale camera and on X-Y translation stages with sub-micrometric positioning precision. An in-house-written software, CELLView [6], allows the remote control of the irradiation protocol. The cell holder is scanned under the microscope, the cells are identified individually by experienced operator eyes and their coordinates are logged; afterwards the sample is moved from the microscope to the beam and cells are irradiated automatically. Mammalian cells (specifically, Chinese hamster V79 cells) are seeded at an appropriate density and grown as a monolayer on a mylar surface used as bottom of the specially designed holder, having as cover an other mylar foil and allowing the cell culture to be in wet and sterile conditions as well as the sample to be transparent for the observation under the microscope and for the transmission of ions during irradiations. Manual recognition of unstained cells in bright-field is a time consuming task that restricts the irradiation cell troughput. Recently, to overcome such a restriction, an automated cell recognition system for single-cell irradiations [12] has been continued with a view to its insertion in the CELLView software.

Many technical problems have been faced to overcome the complexity of the images to be analyzed. The unstained cells have to be discriminated in an inhomogeneous background, among many disturbing bodies mainly due to the mylar surface roughness and culture medium. Additionally, cells can have various shapes depending on how they attach on the surface, which phase of the cell cycle they are in and on cell density, thus making the detection task more difficult.

When cells are stained and scored with a fluorescent microscope, a simple binarisation with morphological processing is sufficient to obtain marker points that show a cell reference. This geometric approach [10] is very fast and reliable when the background is easily separable from objects. In the case considered here, unstained V79 cells, differ not only in shape but also in texture. Moreover, the sample images contain additional objects from the mylar structure or from the culture medium, which make tricking the recognition. Therefore, in this case, geometric methods are not appropriate and an approach based on texture segmentation with supervised learning has been conceived.

2. Cell body detection

To discriminate textures on the cell images, three classes of regions of interest were defined by spots: background, cell edge and cell inner body part. Regions of interest (ROIs) from these classes were extracted manually on learning images. They were classified by an expert operator (see Fig. 1). Feature vectors are constructed by cal-

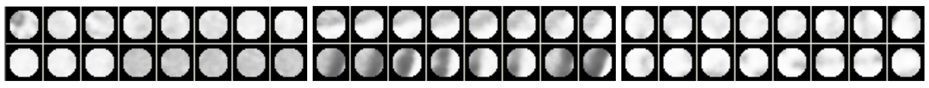


Fig. 1. Example regions of interest extracted from learning images.

culation of texture features from each manually extracted region. Selected classifier is trained to a sufficient level using previously combined learning vectors.

Detection of the cell body (Fig. 2) is performed by classification of overlapping regions of interest from the captured image. For each automatically obtained spot, texture features are calculated, a feature vector is constructed and it is presented to the classifier. Information from classifier is tested against all defined classes and new pixel intensity depends on the classification result, thus every new pixel corresponds to one region in the original image. After preliminary classification of pixels, a new simplified picture with smaller resolution is obtained.

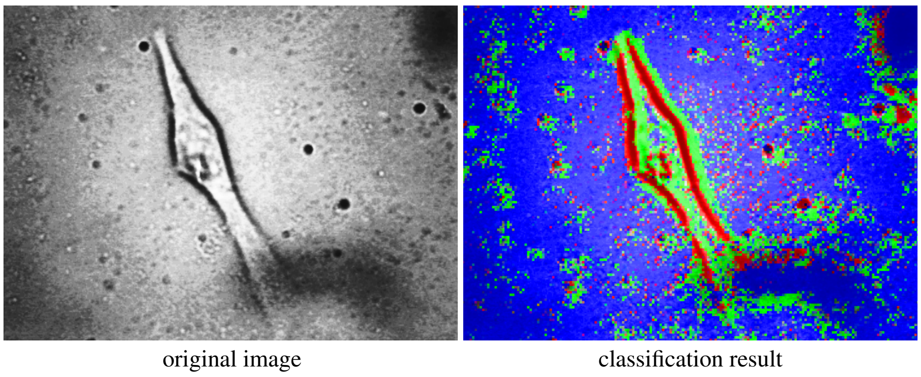


Fig. 2. Result of the classification of overlapping regions of interest by a neural network (cell seeded on cell culture plastic)

In particular, features from normalized histogram, the absolute value of gradient features; and second order statistics: the co-occurrence matrix [16] and run-length matrix [13] features for selected angles are calculated. Additional two feature vectors were evaluated: features obtained after 2D wavelet decomposition using Daubechies 4 wavelet; also Fourier and wavelet features calculated from polar coordinates: from extracted vectors discrete Fourier or wavelet coefficients are calculated and for each coefficient, standard deviation from all vectors is obtained. This creates an energy feature vector that is rotation invariant (see Fig. 3). The computation time of these two additional feature vectors was not satisfactory to fit into our experimental requirements.

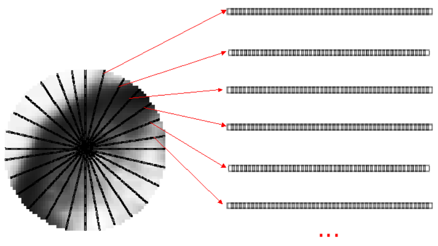


Fig. 3. Texture features extracted from polar coordinates.

Furthermore, captured images of cells were normalized and for each extracted region of interest a raw pixel data was also used to construct feature vectors.

Singular Value Decomposition, SVD [9] was performed to reveal minimum number of dimensions required to represent a feature vector. Often multi-dimensional data may be represented approximately in fewer dimensions due to redundancies in data. For practical purposes, only first K principle components were considered, giving new feature vectors with reduced dimensionality, but still keeping most of their variance.

After the calculation of SVD depending on the aim of executed classification tests, the Independent Components Analysis was additionally performed. ICA seeks directions that are most statistically independent. Number of features is not decreased, but the coordinate system is modified giving a clustered structure. The FastICA [7] algorithm was used to separate independent components.

Computed texture features of the cell body, edge and the background (with corresponding class number) were presented as input vectors to the classifier and it was trainined to a sufficient error level. Three classifiers and one cascade of classifiers were considered in this research.

- Artificial feed-forward neural network with back-propagation learning [11], to achieve training in parallel manner, the input dataset was split among worker nodes and each node calculated only its part of the contribution to overall gradient. Learning was stopped when learning error dropped below 10^{-4}.
- Simplified Fuzzy ARTMAP [8], a slow learning (with small β parameter) was performed and it was stopped when all training vectors presented to the SFAM were classified correctly.
- Support Vector Machines, SVM [14][2], the Radial Basis Function (RBF) kernel was selected in these tests: $K(x_i, x_j) = exp(-\gamma||x_i - x_j||^2), \gamma > 0$.
- Cascade of Support Vector Machines: each node of the SVM cascade was learned using only some part of the training dataset. First layer of SVM nodes obtained normalized input vector directly, other layers were constructed in such a way, that input for it were distances from hyperplanes of the input vector presented to the previous layer.

Obtained new image can be classified once again using different classifier and another database of learning datasets and classes. This approach creates additional layer of classification and can be interpreted as second level of resolution, where first classifier detects small details and another one detects groupped details into one more recognizable object. However, for this technique one needs to have images in very high resolution and the recognition time is prolonged considerably.

3. Cell shape recognition

The simplified image is morphologically processed and potential cell markers for the segmentation algorithm are acquired. Watershed with markers [1] on the original gradient image is applied (see Fig. 4).

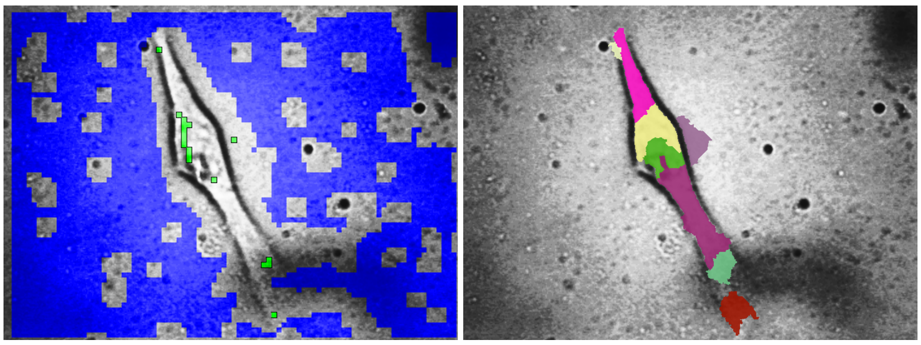


Fig. 4. Example segmentation (cell on plastic).

Segmented objects were extracted from the image and their shape features were calculated: height, width, perimeter, volume, major and minor axis length, roundness of the object, equivalent diameter and Fourier shape descriptors. Additionally, all texture features within segmented object are computed. For all feature vectors from learning set also dimensionality reduction by calculation of the SVD was performed and subsequently the ICA. Objects from learning images were qualified by the operator, and, when necessary, over-segmented objects were joined manually. Their normalized shape and texture features were presented to the additional classifier and supervised training was performed.

During recognition, shape features of combinations of objects connected into one region are calculated. The obtained combination is considered as detected cell when a new region constructed from joined objects is classified as a "cell body" by the classifier and volume of that region is higher than other combinations classified as a "cell body". In such a case, all connected objects from this combination are extracted from the original region and the operation is repeated until all remaining combinations are classified as "not cell body". For practical reasons, not all combinations are considered for segments joining, and before any calculation of shape features, they are sorted descending by their volume and some criteria is applied to filter out objects and combinations that are for sure not cells.

```
Input: segments, criterion
find continous regions in segments
C ← combinations of segments joined into every continous region
remove regions from C where joined segments do not satisfy criterion
sort regions in C descending by the volume
foreach Region ∈ C do
    if classify(Region) = cell then
        push result(Region)
        remove regions from C, where ∃ segments also in Region
    end
end
```

Algorithm 1: Algorithm for cells recognition from oversegmented objects

The task of discrimination which segments have to be joined depends on the classifier's accuracy. This approach was preferred rather than calculation of segments similarity, because of presence of the mylar grain that can be located also on the cell body. Moreover, proper recognition of favoured shape helps in correct segments joining.

For each recognized cell the largest inscribed circle (Fig. 5) is calculated using the Voronoi diagram [15]. Circle centers define the point (X,Y) with respect to an

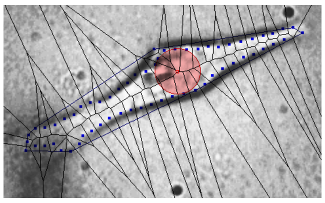


Fig. 5. Largest inscribed circle (cell on plastic).

appropriate reference marker, where the irradiation should be performed.

4. Experimental results

Main calculations are executed on the cluster of machines. Every worker node calculates only its own part of the whole data set. During on-line recognition, ROIs and shapes are spread among worker nodes for parallel classification. Average time to perform textures classification of 15470 ROIs with 20x20 pixels window size was 9.5 seconds for INFN-LNL 10 dual-core machines (Intel Xeon 2.4GHz, 19 workers).

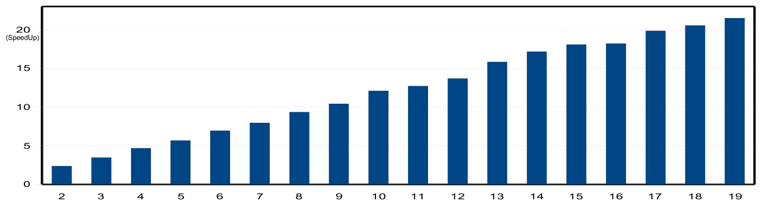


Fig. 6. Average speed up time of textures classification for different number of worker nodes $SpeedUp = \frac{Time_{sequential}}{Time(P)_{parallel}}$.

From performed experiments appeared that texture of the cell edge is very hard to detect and the best efficiency was obtained for the SVM classifier, without applying any changes in the coordinate system. It was very small, only 87%, considering that the cell inner part achieved 97% of properly classified spots. The best accuracy for the cell shape classification (95%) was reached for all shape and texture features, after applying dimensionality reduction. For the SVM decreased training dataset sometimes achieved better discrimination. This can be explained by overfitting of the classifier. The proposed SVM cascade had slightly lower accuracy than pure SVM in all performed experiments. The neural network had precision below the SVM (around 3% below) and the worst discrimination was obtained after applying the ARTMAP classifier (no more than 82%).

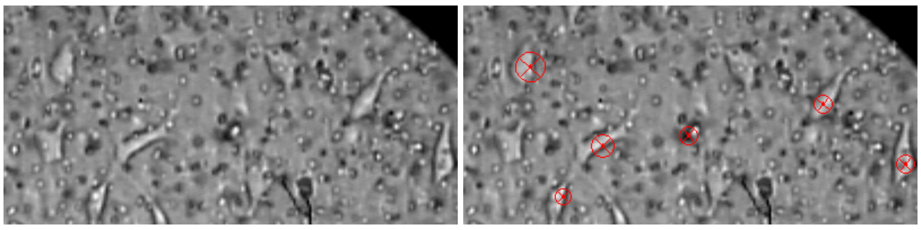


Fig. 7. Example recognition of unstained V79 cells placed between two mylar foils.

5. Conclusions

Unstained cells are very hard to recognise, especially when confounding objects and not clean background feature the acquired image. Supervised training of the classifier helps to qualify small parts of the image, providing additional information for the segmentation algorithm. At present, the classification is not optimized and the operator has to perform additional revision of cells coordinates. Further improvements are in progress.

Acknowledgements

This work was partially supported by the CELLION project MRTN-CT-2003-503923. The authors acknowledge: M. Mackey (Iowa University, USA) for his support and the interesting discussions, as well as for the hospitality (of M.S.) in his Institute; M. Biasotto (INFN-LNL Computing Dept., Italy) for his support and for making possible access and use of LNL cluster machine; Z. Stachura and J. Lekki (IFJ-PAN, Krak"ow, Poland); the Biologists of LNL Radiobiology Lab, V. De Nadal and D. Guryev (CELLION post-doc fellow), for the preparation of the culture cells used in this work and for the useful discussions on the various biological aspects.

References

[1] S. Beucher and F. Meyer: Mathematical Morphology in Image Processing, chapter 12, pages 433–481. Marcel Dekker, 1993.

[2] Chih C. Chang and Chih J. Lin: LIBSVM: a library for support vector machines, 2001.

[3] S. Gerardi: A comparative review of the charged particle microbeam facilities, Radiat. Prot. Dos., 122:285–291, 2006.

[4] S. Gerardi, G. Galeazzi, and R. Cherubini: A microcollimated ion beam facility for investigations of the effects of low-dose radiation, Radiation Research, 164(4):586–590, 2005 (and references therein).

[5] S. Gerardi, G. Galeazzi, and R. Cherubini: Single ion microbeam as a tool for low-dose radiation effects investigations, Journal of Physics: Conference Series, 41:282–287, 2006.

[6] S. Gerardi and E. Tonini: CELLView: a software control system for sample movement, single-cell visualization and micropositioning at the LNL horizontal single-ion microbeam facility, LNL Annual Report 2002, page 65, 2003.

[7] A. Hyvärinen and E. Oja: Independent component analysis: algorithms and applications, Neural Netw., 13(4-5):411–430, 2000.

[8] Mohammad-Taghi, Vakil-Baghmisheh, and Nikola Pavešić: A fast Simplified Fuzzy ARTMAP network, Neural Process. Lett., 17(3):273–316, 2003.

[9] R. Philips, L. Watson, R. Wynne, and C. Blinn: Feature reduction using singular value decomposition for the IGSCR Hybrid Classifier, In The 2007 International Conference on Scientific Computing, 2007.

[10] S. Raman, Bahram Parvin, C. Maxwell, and Mary Helen Barcellos-Hoff: Geometric approach to segmentation and protein localization in cell cultured assays, In George Bebis, Richard D. Boyle, Darko Koracin, and Bahram Parvin, editors, ISVC, volume 3804 of Lecture Notes in Computer Science, pages 427–436. Springer, 2005.

[11] D. E. Rumelhart, E. E. Hinton, and R. J. Williams: Learning representations by back-propagating errors, Nature, 323:533, Oct 1986.

[12] M. Skoczylas, R. Cherubini, and S. Gerardi: Automatic unstained cells recognition for single-cells irradiations, LNL Annual Report 2006, page 61, 2007.

[13] Fumiaki Tomita and Saburo Tsuji: Computer Analysis of Visual Textures, Kluwer Academic Publishers, Norwell, MA, USA, 1990.

[14] V. Vapnik, S. Golowich, and A. Smola: Support vector method for function approximation, regression estimation, and signal processing, Neural Information Processing Systems, 9, 1997.

[15] Georgy Voronoi: Nouvelles applications des paramètres continus à la théorie des formes quadratiques, Journal für die Reine und Angewandte Mathematik, pages 97–178, 1907.

[16] A. Zizzari, B. Michaelis, and G. Gademann: Optimal feature functions on co-occurrence matrix and applications in tumor recognition, In Applied Simulation and Modelling, 2002.

AUTOMATYCZNA DETEKCJA ŻYWYCH KOMÓREK NIEBARWIONYCH W OBRAZACH WYKONANYCH MIKROSKOPEM FAZOWO-KONTRASTOWYM

Streszczenie: Precyzyjne napromieniowywanie żywych komórek biologicznych z użyciem znanej liczby jonów (z dokładnością do pojedynczej cząstki) pozwala na badanie efektów niskiej dawki promieniowania oraz dostarcza unikalnych danych służących ocenie ryzyka wystąpienia choroby nowotworowej jako konsekwencji ekspozycji na niskie dawki promieniowania jonizującego. Komórki ssaka (dokładniej chomika chińskiego, komórki V79) są hodowane w jednej warstwie na powierzchni folii wykonanej z cienkiego mylaru, użytej jako dolna część specjalnie do tego celu zaprojektowanej szalki, posiadającej jako przykrycie kolejną, wierzchnią warstwę folii, umożliwiając komórkom przebywanie w sterylnym otoczeniu. Płytka z komórkami jest skanowana pod mikroskopem, komórki są identyfikowane przez operatora oraz zapisywane są ich współrzędne; po tej procedurze próbka jest przesuwana z mikroskopu pod wiązkę i komórki są automatycznie napromieniowywane. Ręczne rozpoznawanie komórek niebarwionych w obrazach uzyskanych mikroskopem fazowo-kontrastowym jest bardzo czasochłonną procedurą, dlatego został zaprojektowany oraz zaimplementowany równoległy algorytm, aby przyspieszyć ten krok protokołu radiacji oraz by zwiększyć ilość komórek które mogą zostać napromieniowane w pojedynczym eksperymencie, którego czas powinien być skrócony do minimum. Napotkano wiele problemów w przezwyciężeniu trudności analizy tego rodzaju obrazów. Komórki niebarwione muszą zostać rozróżnione na niejednorodnym tle, pośród wielu dodatkowych obiektów, pochodzących głównie ze struktury folii mylar oraz środowiska w którym żyją komórki. Dodatkowo, kształty komórek są bardzo zróżnicowane, w zależności od sposobu w jaki przyczepiają się do powierzchni, w jakim cyklu komórkowym aktualnie przebywają oraz ich gęstości, powodując iż rozwiązanie problemu rozpoznawania ich jest trudnym zadaniem.

Słowa kluczowe: rozpoznawanie komórek niebarwionych, mikrowiązka jonowa, analiza obrazu

Artykuł zrealizowano w ramach grantu MRTN-CT-2003-503923.

ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2009
Informatyka – Zeszyt 4

Marta K. Smolińska[1], Zenon A. Sosnowski[1]

PARALLEL FUZZY CLUSTERING FOR LINGUISTIC SUMMARIES

Abstract: The linguistic summaries have the associated truth value so they can be used as predicates. We use summaries of the form "most objects in population P are similar to o_i" to find typical values in population P. Then typical values are used in fuzzy clustering algorithm. Disadvantage of this algorithm is its complexity. For the purpose of processing the huge number of data, we decided to use parallel computing mechanism to implement this algorithm, and run it on the cluster machine. We use MPI (Message Passing Interface) to communicate between processes, which work on different processors. This paper presents this parallel algorithm and some results of experiments.

Keywords: linguistic summary, fuzzy clustering, parallel computing

1. Introduction

In the preceding article [1] we presented the algorithm of clustering objects in object-oriented database, based on linguistic summaries, according to Yager's approach [2,3]. We also introduced the certain modification of Yager's algorithm which improves the clustering but extremely increases complexity, which is high, in this algorithm without modification. For the purpose of processing the huge number of data, we decided to use parallel computing mechanism to implement this algorithm, and use it on the cluster machine. We use MPI (Message Passing Interface) to communicate between processes, which work on different processors. This paper presents this parallel algorithm and some results of experiments.

The paper is organised as follows. Section 2. describes the linguistic summary. In Section 3 we describe typical values, and in Section 4. their use for searching typical clusters is presented. In Section 5. we introduce parallel version of clustering algorithm. Then in Section 6. we present the results of experiments. Finally, the conclusion will follow in Section 7.

[1] Faculty of Computer Science, Bialystok Technical University, Białystok

2. Linguistic summary

Rasmussen and Yager in [2,3] propose linguistic summary of the form "Q objects in P are S". In this paper we will use notation $Summary(Q,P,S)$ to express that summary. For example linguistic summary can look as follows *"most people are tall" (Summary(most, people, tall)), "few tall people are light" (Summary(few, tall people, light)*.

Q is the quantity in agreement (a linguistic quantifier as most, few etc.), P is a data collection and S is called the summarizer (e.g. young, tall), which very often is a fuzzy predicate.

The truth value $\tau[0,1]$, called the measure of validity is associated with a linguistic summary. It provides an indication of how compatible the linguistic summary with the population is. As a population we mean a fuzzy set of objects. Each object o in a population P has a degree of membership to P - $\mu_P(o)$ (we will also use symbol $o.\mu$).

Measure of validity is calculated as the grade of membership of the proportion of objects in P that satisfy S (eq. 1).

$$\tau = \mu_Q\left(\frac{card_f(S \cap P)}{card_f(P)}\right) \tag{1}$$

$\cap$ is defined as the minimum aggregation. $card_f$ is the fuzzy cardinality of a fuzzy subset and is defined by equation 2

$$card_f(P) = \sum_{o_i \in P} \mu_P(o_i) \tag{2}$$

Complexity of calculating the true degree τ of a linguistic summary is $O(N)$, where N is the number of objects in P.

3. Typicality - typical values

Typicality of an object tells as how much the object is typical in the population. It is calculated by means of linguistic summary, which like a fuzzy predicate has a truth value. So we can use it as a predicate. Value of typicality $t(o_i)$ (we will use also syntax $o_i.t$) of object o_i is the minimum of membership degree of o_i and measure of validity of summary: "most objects in P are similar to o_i" like in eq. 3

$$t(o_i) = o_i.t = min(\mu(o_i), Summary(most, P, \text{Similar\_to}(o_i))) \tag{3}$$

In order to query about typical objects in P, we can establish a new population *typical*, which will consist of objects from P with associated typicality. α-cut can be used to cut off objects with typicality lower than α. Complexity of calculating typicality for one object is $O(N)$, where N is the number of objects in P. So creating population *typical* (calculating all objects in population) has a complexity $O(N^2)$.

4. Fuzzy clustering algorithm

This algorithm is based on Rasmussen approach [2]. Algorithm 1 presents an idea of finding typical clusters using linguistic summaries. In [1] we modified this algorithm by adding step 4. This gives better clustering results but increases complexity from $O(N^2)$ to $O(N^3)$. Practically this complexity depends on the number and size of the obtained clusters.

Algorithm 1 TypicalClusters(P, Q, Sim)

```
Require: $P-$ population, $Q-$ function (pointer to function) of fuzzy quantifier, $Sim(o_i, o_j)$ - similarity
    function $o_i, o_j \in P$
 1: For each object $o$ in population count its typicality $o.t$. Add them all to the population typical,
    remember their typicality $o.t$ and membership degree to the original population $o.\mu$.
 2: Let the most typical object $o'$ $(\forall o \in typical : o'.t \geq o.t)$ be a seed for a new cluster $C_i$.
 3: Find objects $o_j$ from typical that are close to any object of $C_i$, that is $\exists o \in C_i$: $Similarity(o_j, o) \geq$
    $\alpha$. Add them to the cluster $C_i$ and remove from typical. Continue until no more objects are added.
    The cluster $C_i$ will then constitute a typical cluster.
 4: For each object $o$ in typical count its typicality $o.t$, among objects remaining in typical. In this
    calculations we use membership degree that objects had in original population $o.\mu$.
 5: To find next cluster repeat from step 2, as long as there are objects in typical, and the most typical
    object $o'$ has a degree of typicality greater or equal than $\beta$ $(o'.t \geq \beta)$.
 6: return Clusters $C_0 \ldots C_n$ {$n$ - number of created clusters}
```

Clustering Quality

We tested our and Rassmusen algorithms on database formed from pixels of an image. To create the resultant image, we assign colour of the first object in cluster (which was added to cluster as first) to all pixels from this cluster. We run this algorithm with factor $\beta = 0$ to cluster all pixels. To compare results with original image we use Euclidean distance metric between colours in RGB space eq. 4.

$$D(o,r) = \sum_{i,j} \sqrt{(o[i,j].R - r[i,j].R)^2 + (o[i,j].G - r[i,j].G)^2 + (o[i,j].B - r[i,j].B)^2} \quad (4)$$

where o and r are original and result image respectively. This metric is chosen for its computational speed and simplicity.

Rassmusen's algorithm created 305 clusters and the difference D between result and original image was equal to 13 745 431. Modified version created 301 clusters and D was equal to 5 605 994.5 . There is less colours (less created clusters) in the result image and difference from original is over 2 times less than in algorithm without modification.

5. Parallel Fuzzy clustering algorithm

This parallel algorithm was implemented and tested on populations with the large amount of data. In our parallel implementation we use MPI (Message Passing Interface). We execute our algorithm on many processors. One process on each processor.

We have one master process - "special process designated to manage the pool of available tasks" and many slaves - "processes that depend on the master to obtain work" [4]. Each process is executed on different processor of cluster machine.

In our algorithm there are two situations.

First: when we can divide work among processes and we know that it will take the same time (the same number of computations). This is when we calculate typicality for objects of population. Then we use static mapping of tasks onto processes. Each process calculates the part of objects.

Second situation: when we check if object is similar to cluster. Object is similar to the cluster, if it is similar, to one or more elements of cluster. In optimistic situation, when object is similar to the first object of the cluster, the computation of similarity is done only once. In pessimistic situation - when object is not similar to the cluster, we have to calculate similarity as many as cluster size. To obtain load balancing we use dynamic mapping of tasks to processors. The master tells slaves which object they have to check of similarity to the cluster. When slave process returns result to master, then it receives another task if there are any.

Message in MPI has a tag. We use it to inform other processes what message is sent. In many situations slaves cannot establish what message will be received. So it checks (with MPI_Probe) what tag the message has, and does adequate operations e.g. allocates memory and then receives message.

The idea of our parallel algorithm is as follows. (Algorithms 2 and 3 present in detail master and slave functions. As a function $typicality(o,P)$ we mean $typicality(o,P) = min(\mu(o), Summary(most, P, \text{Similar\_to}(o))))$

In our case each process needs all the objects, because computing typicality of one object requires all other objects. Master packs the population P, and sends it to all other processes - slaves. The slaves receive packed population and unpack it. So each process has a copy of data. Then slaves calculate typicality of the part of the population - each slave different part. Processes in communicator have identifiers from 0 to $p-1$ where p is number of processors. In our case process with identifier 0 becomes master. If N is the population size (number of objects), process with identifier p_{id} considers objects starting from the object with index equal to $\frac{N}{p-1} * (p_{id} - 1)$ ending with index equal to $(\frac{N}{p-1} * p_{id}) - 1$. When N is not divisible by $(p-1)$ then master calculates the rest of objects with indices from $N - (N \bmod (p-1))$ to $N-1$ (indices of the population are zero-based). Then each slave sends results to master, which accumulates them and sends back to all slaves. So each slave can create copy of the population $typical$.

All processes create new cluster containing one object - the most typical. Master saves all clusters in an array. Slaves do not need to keep all clusters. They use only one actually created cluster.

In the next step, the master sends to each slave index of the object, it have to calculate, if that object is similar to the cluster, or not. Then slave sends back the information, and if there are any other objects, it receives another index. This is repeated until all objects are checked.

Master sends to all slaves indices of objects that have to be added to cluster, and removed from $typical$.

Searching is continued as long as there were any object added to cluster.

If no more objects were added, each slave clears its cluster. All the processes compute typicality of their part of objects that remain in $typical$. Master receives it and accumulates, creates next cluster containing most typical object and so on. This is continued until there are any objects in $typical$ and most typical object has a typicality not less than β.

In this algorithm similarity between each two objects is calculated many times. We had an idea of creating similarity matrix of objects. Such a solution is good with small populations. In the case when the population size is large, similarity matrix is too big to fit in memory. For example: if size of the population is 250 000 then similarity matrix takes the place of $250\ 000^2 * \text{sizeof}(float) = 250\ 000^2 * 4$ bytes. It is over 232 GB. At this moment this is too large to fit into memory. On the other hand keeping it on hard disk is not profitable because of disk access time.

Algorithm 2 TypicalClustersMaster(P, p, Q, Sim, A, B)

```
Require: P− population, p− number of processes ($p \geq 2$), Q− pointer to function of fuzzy quantifier,
    $Sim(o_i, o_j)$ - similarity function $o_i, o_j \in P$, A, B - functions counting value of factors $\alpha$ and $\beta$
 1: pack population P and send it to all slaves
 2: if P.size mod $(p-1) > 0$ then
 3:     for $i = N - (N \bmod (p-1))$ to $N-1$ do
 4:         $o_i.t = typicality(o_i, P)$   $typical.Add(o_i)$
 5:     end for
 6: end if
 7: receive from slaves and save in typical typicality of objects they calculated
 8: send typicality of all objects to all slaves
 9: find $o'$ the most typical object {the object with the highest degree of typicality}
10: $\alpha = A(o'.t)$   $\beta = B(o'.t)$   $k = 0$
11: while $typical.Size > 0$ and $o'.t \geq \beta$ do
12:     create new cluster $C_k$
13:     $C_k.Add(o')$   $typical.Remove(o')$
14:     repeat
15:         send to each slave a consecutive index of object {Slave counts similarity of this object to the
            cluster $C_k$}
16:         while any slave count similarity do
17:             receive from any slave $s_x$ similarity of object $o_i$ and if it is similar save i in the array Inds
18:             if there are indices, that wasn't send then
19:                 send consecutive index i to slave $s_x$.
20:             end if
21:         end while
22:         send to all slaves array Inds.
23:         for all objects $o_i : i \in Inds$ do
24:             $C_k.Add(o_i)$   $typical.Remove(o_i)$
25:         end for
26:     until $Tab.Size > 0${there where any objects added to cluster $C_k$}
27:     if $typical.Size < p$ then
28:         send message with tag=11 to redundant slaves {Slave exits.}
29:         $p = p - typical.Size$ send p to other slaves
30:     end if
31:     $N = typical.size$
32:     if N mod $(p-1) > 0$ then
33:         for $i = N - (N \bmod (p-1))$ to N-1 do
34:             $o_i.t = typicality(o_i, typical)$
35:         end for
36:     end if
37:     receive from slaves and save in typical typicality of objects they calculated
38:     send typicality of all objects to all slaves
39:     find $o'$ the most typical object
40:     $\alpha = A(o'.t)$   $\beta = B(o'.t)$   $k = k + 1$;
41:     send $\alpha$ and index of $o'$ to all slaves
42: end while
43: return clusters $C_0 \dots C_n$
```

Algorithm 3 TypicalClustersSlave$(p, p_i d, Q, Sim)$

```
Require: $p-$ number of processes ($p \geq 2$), $p_{id}$ - rank of this process, $Q-$ pointer to function of quantity
    in agreement,$Sim(o_i, o_j)$ - similarity function $o_i, o_j \in O$
 1: receive and unpack population $original$
 2: $N = original.Size$
 3: for all $o_i$, such that $\frac{N}{p-1} * (p_{id} - 1) \leq i \leq \frac{N}{(p-1)} * p_{id}$ do
 4:     $T[i] = typicality(o_i, original)$
 5: end for
 6: send filled part of $T$ to master
 7: receive from master index $m$ of the most typical object $o' = o_m$
 8: receive from master array $T$ {typicality of all objects}
 9: create new population $typical$ from objects of $original$ and typicality from array $T$
10: create new cluster $C$ with most typical object $o'$
11: remove $o'$ from $typical$
12: repeat
13:     test for a message from master
14:     if message.tag = 11 then
15:         exit;
16:     else if message.tag = 4 then
17:         receive $\alpha$ from Master
18:     else if message.tag = 6 then
19:         receive index $i$ from master
20:         check if $o_i$ is similar to the cluster $C$ and send similarity to master
21:     else if message.tag = 7 then
22:         receive array $Inds$ from master
23:         if $Inds.size > 0$ then
24:             for all objects $o_i : i \in Inds$ do
25:                 $C.Add(o_i)$;   $typical.Remove(o_i)$;
26:             end for
27:         else {there are no objects similar to cluster.}
28:             if $typical.size < p - 1$ then
29:                 receive $p$ from master or exit if message.tag = 11
30:             end if
31:         end if
32:         N=typical.Size
33:         for all $o_i \in typical$, such that $\frac{N}{p-1} * (p_{id} - 1) \leq i \leq \frac{N}{(p-1)} * p_{id}$ do
34:             $T[i] = typicality(o_i, typical)$
35:         end for
36:         send filled part of $T$ to master
37:         receive from master index $m$ of the most typical object $o' = o_m$
38:         receive from master array $T$ {typicality of all objects in $typical$}
39:         for all $o_i \in typical$ do
40:             $o_i.t = T[i]$
41:         end for
42:         $C.clear$   $C.Add(o')$   $typical.Remove(o')$
43:     end if
44: until message.tag $\neq$ 11
```

6. Results of experiments

We tested our algorithm on Compute Cluster - ”mordor2” at Faculty of Computer Science of Bialystok Technical University. There are 16 compute nodes, each of 2 quad-core processors Intel Xeon X5355 (2660 MHz) with 4GB RAM. The program is implemented in C++, and we use Intel compiler.

We performed three experiments with three various population size. In the first experiment, there was 65 536 objects (44 192 unique values of attribute on witch similarity function was defined) and 190 clusters were created. In the second 301 clusters from population of 262 144 objects (69 997 unique values). The last experiment was performed on population 1 064 000 objects. This time only 28 clusters were created , because there was only 254 unique values of attribute. We used factors: $\alpha_p = 75\%$ of the most typical value ($\alpha = o'.t * 75\%$) and $\beta = 0$ to cluster all the objects. Figures 1, 2 ,3 show speedup of our parallel algorithm for those three experiments and Table 1 contains time in seconds of performing each of this three experiments.

In all figures p is the processors number, $T_1(p)$ - is time of processing first experiment on p processors, $T_2(p)$ and $T_3(p)$ second and third experiment accordingly. $T(1)$ - is the time of running program sequentially on one processor. Speedup of algorithm is calculated by formula in eq. 5.

$$S(p) = \frac{T(1)}{T(p)} \tag{5}$$

Analysing the charts we can say that there is no speedup for 2 processors because of algorithm architecture. One of two processes is master and it doesn’t take part in calculations. From 5 processors to 20 processors (fig. 2) we can say that speedup is close to linear speedup. It grows slower for greater number of processes as a consequence of Amdahl’s law. It starts to saturate earlier (with less number of processors) for smaller population (fig. 1 and later for greater population (fig. 3). Like in fig. 3 the chart locally drops. It may be caused by load imbalance - there are situations when master participates in calculations or not, depending on the number of processors and actual number of objects. It is noteless with smaller population (fig. 1). The time of running program on one processor in third experiment is 73 638s. that is over 20 hours, while on 48 processors 1 760.04s - somewhat less than half an hour.

7. Conclusion

The parallel version of this algorithm allows to deal with large data bases in acceptable time. The more objects we have, the more processors we can use without

Table 1. Time of performing experiments

p	$T_1(p)$	$T_2(p)$	$T_3(p)$	p	$T_1(p)$	$T_2(p)$	$T_3(p)$
1	536	8833	104 066	25	28.024	392.796	2862.57
2	542.771	8827.04	111943	26	26.6639	398.204	3086.98
3	440.099	6917.32	76519.8	27	26.1264	384.812	2973.57
4	369.37	5938.48	41201	28	25.2003	375.33	2885.53
5	294.841	2232.53	16613.9	29	24.4827	341.483	2482.13
6	160.135	1786.62	13615	30	23.8073	331.278	2425.07
7	93.7401	1495.43	11613.9	31	23.3445	341.783	2724.54
8	81.7998	1372.49	11135.7	32	22.8769	330.902	2649.3
9	72.1456	1104.92	24866.1	33	22.1609	321.069	2430.05
10	63.8581	991.029	14712.6	34	21.681	310.859	2358.91
11	57.6498	899.491	6788.27	35	21.3022	304.901	2303.22
12	52.7288	820.887	6253.87	36	20.7538	279.391	3606.57
13	48.8878	761.468	5829.14	37	20.7321	305.128	2282.51
14	45.498	746.249	5958.49	38	20.3226	299.343	2219.04
15	43.1952	691.14	9235.31	39	20.0423	293.134	2162.86
16	40.291	684.79	5434.6	40	19.6307	284.499	2113.97
17	38.0155	572.078	4291.65	41	19.1846	263.208	1953.81
18	36.14	541.105	4086.26	42	18.8672	257.352	1910.82
19	34.937	510.345	3774.39	43	18.8057	242.463	1965.2
20	33.3961	484.716	6627.98	44	18.4836	260.475	1921.94
21	31.7411	493.387	3862.84	45	18.3377	232.151	1879.08
22	30.6009	495.573	3874.26	46	18.1134	250.914	1834.79
23	29.4827	476.159	3699.19	47	17.8429	246.288	1794.08
24	28.4549	456.875	3556.91	48	17.6971	242.1	1760.04

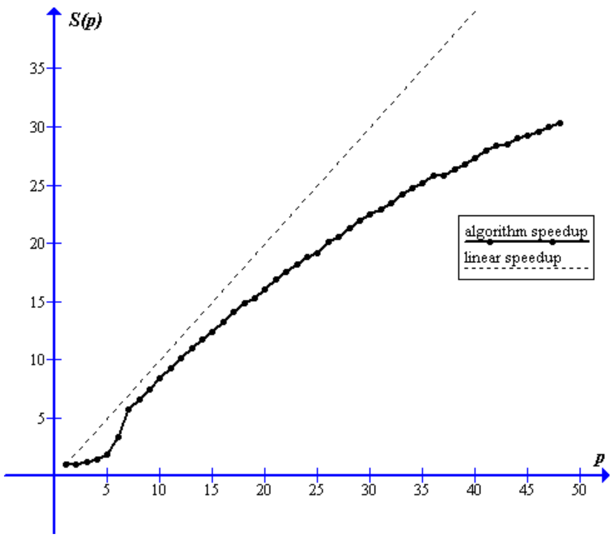


Fig. 1. Algorithm speedup for population of 65 536 objects

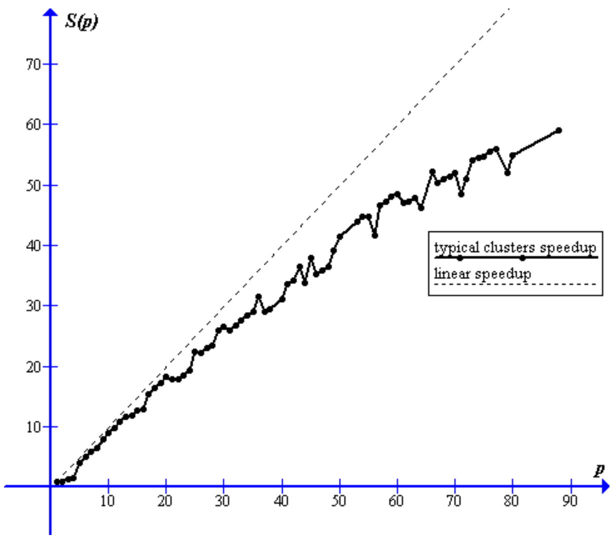


Fig. 2. Algorithm speedup for population of 262 144 objects

efficiency degradation. Our algorithm is not perfect. We use only blocking communication operations and there is moment of poor load balancing (master should more participate in computation). There are also messages that master sends to all pro-

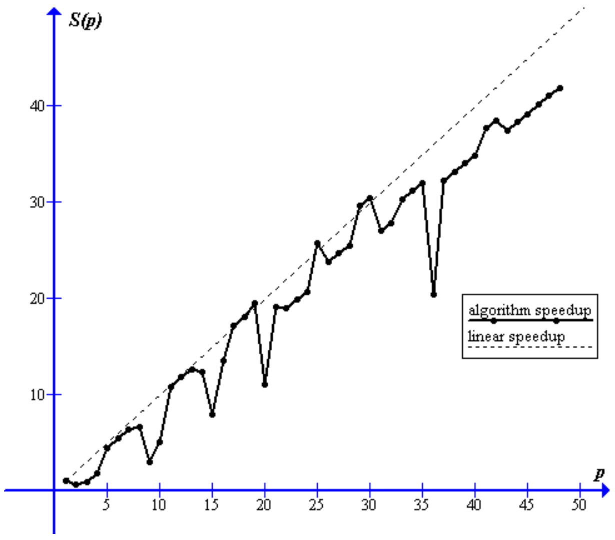


Fig. 3. Algorithm speedup for population of 1 064 000 objects

cesses. It would be better to use collective communications operations. We have not implemented those yet.

We didn't make parallel linguistic summaries, but only clustering algorithm, because complexity of the former is $O(N)$. Object-oriented databases, that are subject of our interest, allow creating object attributes that are collections of other objects or other data like multimedia. In that situation, predicate defined on such an attribute or group of attributes may be very time consuming. Next we plan to implement parallel linguistic summaries to deal with such databases.

References

[1] Smolińska M. Sosnowski Z.: Linguistic Summaries with Fuzzy Clustering, Zeszyty Naukowe Politechniki Białostockiej. Informatyka Nr 2 (2007), s.141-154 (in Polish)

[2] Rasmussen D.: Application of the Fuzzy Query Language - Summary SQL, DATALOGISKE SKRIFTER, Roskilde University, 1997.

[3] Rasmussen D., Yager R.: Introduction of Fuzzy Characteristic Rules by Typical Values, DATALOGISKE SKRIFTER, Roskilde University, 1997.

[4] Grama A., Gupta A., Karypis G., Kumar V.: Introduction to Parallel Computing, Second Edition, Addison Wesley, 2003.

Marta K. Smolińska, Zenon A. Sosnowski

PODSUMOWANIA LINGWISTYCZNE Z RÓWNOLEGŁYM GRUPOWANIEM ROZMYTYM

Streszczenie: Z podsumowaniem lingwistycznym, jak i z predykatem rozmytym związana jest wartość prawdy. Możemy więc podsumowań lingwistycznych używać jako predykatów rozmytych. Podsumowanie postaci ”większość obiektów w populacji *P* jest podobna do obiektu o_i wykorzystać możemy do znajdowania typowych wartości w populacji *P*, które to wykorzystuje rozmyty algorytm grupujący. Wadą tego algorytmu jest jego duża złożoność obliczeniowa. W celu przetwarzania dużej liczby danych zaimplementowaliśmy ten algorytm równolegle, korzystając ze standardu MPI do komunikacji między procesami działającymi na różnych procesorach. W tej pracy przedstawiamy algorytm równoległy i wyniki eksperymentów.

Słowa kluczowe: podsumowania lingwistyczne, grupowanie rozmyte, programowanie równoległe

Artykuł zrealizowano w ramach pracy badawczej S/WI/2/08.

ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2009
Informatyka – Zeszyt 4

Kazimierz Trzęsicki[1]

LOGIC IN FORMAL VERIFICATION OF COMPUTER SYSTEMS. SOME REMARKS.

Abstract: Various logics are applied to specification and verification of both hardware and software systems. The problem with finding of proof is the most important disadvantage of proof-theoretical method. The proof-theoretical method presupposes the axiomatization of the logic. Proprieties of a system can also be checked using a model of the system. A model is constructed with the specification language and checked using automatic model checkers. The model checking application presupposes the decidability of the task.

Keywords: Logic, Verification, Proof-theoretical Method, Model Checking

1. Logic in Computer Science

Connections between logic and computer science (*CS*) are wide-spread and varied. Notions and methods from logic can fruitfully be applied within *CS*. Logic plays the same role in *CS* as the calculus plays in physics. Logic is „the calculus of computer science" [74,16,29].

On the one hand, logic permeates more and more its main areas. On the other hand we may notice that [59, p. 181]:

> Until the invention of the digital computer, there were few applications of formal mathematical logic outside the study of logic itself. In particular, while many logicians investigated alternative proof systems, studied the power of various logics, and formalized the foundations of mathematics, few people used formal logic and formal proofs to analyze the properties of other systems. The lack of applications can be attributed to two considerations: (i) the very formality of formal logic detracts from its clarity as a tool of communication and understanding, and (ii) the "natural" applications of mathematical logic in the pre-digital world were in pure mathematics and there was little interest in the added value of formalization. Both of these considerations

[1] University of Bialystok

changed with the invention of the digital computer. The tedious and precise manipulation of formulas in a formal syntax can be carried out by software operating under the guidance of a user who is generally concerned more with the strategic direction of the proof.

The logical methods are applicable for the design, specification, verification[2] and optimization of programs, program systems and circuits. Logic has a significant role in computer programming. While the connections between modal logic[3] and *CS* may be viewed as nothing more than specific instances, there is something special to them.

In 1974 the British computer scientist Rod M. Burstall first remarked on the possibility of application of modal logic to solve problems of *CS*. The Dynamic Logic of Programs has been invented by Vaughan R. Pratt [81]:

In the spring of 1974 I was teaching a class on the semantics and axiomatics of programming languages. At the suggestion of one of the students, R. Moore, I considered applying modal logic to a formal treatment of a construct due to C. A. R. Hoare, "$p\{a\}q$", which expresses the notion that if p holds before executing program a, then q holds afterwards. Although I was skeptical at first, a weekend with Hughes and Cresswell[4] convinced me that a most harmonious union between modal logic and programs was possible. The union promised

[2] Some authors distinguish between ***Validation*** and ***Verification*** and refer to the overall checking process as V&V. Validation is answering to the question: *Are we trying to make the right thing?*. Verification answer the question: *Have we made what we were trying to make?* In general methodology of sciences the term "verification" denotes establishing correctness. The term "falsification" (or "refutation") is used in meaning: to detect an error. In *CS* "verification" covers both the meanings and refers to the two-sided process of determining whether the system is correct or erroneous.

For Dijkstra [33] the verification problem is distinct from the pleasantness problem which concerns having a specification capturing a system that is truly needed and wanted. Emerson observes that [36, p. 28]:

The pleasantness problem is inherently pre-formal. Nonetheless, it has been found that carefully writing a formal specification (which may be the conjunction of many sub-specifications) is an excellent way to illuminate the murk associated with the pleasantness problem.

[3] The traditional modal logic deals with three 'modes' or 'moods' or 'modalities' of the copula 'to be', namely, *possibility*, *impossibility*, and *necessity*. Related terms, such as *eventually*, *formerly*, *can*, *could*, *might*, *may*, *must*, are treated in a similar way, hence by extension, logics that deals with these terms are also called modal logics.

The basic modal operator $\Box$ (necessarily) is not rigidly defined. Different logics are obtained form different definition of it. Here we are interested in temporal logic that is the modal logic of temporal modalities such as: *always, eventually*.

[4] The book Pratt is talking about is *An Introduction to Modal Logic* [55].

> to be of interest to computer scientists because of the power and mathematical elegance of the treatment. It also seemed likely to interest modal logicians because it made a well-motivated and potentially very fruitful connection between modal logic and Tarski's calculus of binary relations.

This approach was a substantial improvement over the existing approach based on the pre-condition/post-condition mechanism provided by Hoare's logic.[5] Kripke models, the standard semantic structure on which modal languages are interpreted, are nothing but graphs. Graphs are ubiquitous in *CS*.

The connection between the possible worlds of the logician and the internal states of a computer is easily described. In possible world semantics, ϕ is possible in some world w if and only if ϕ is true in some world w' accessible to w. Depending on the properties of the accessibility relation (reflexive, symmetric, and so on), there will be different theorems about possibility and necessity. The accessibility relation of modal logic semantics can thus be understood as the relation between states of a computer under the control of a program such that, beginning in one state, the machine will (in a finite time) be in one of the accessible states. In some programs, for instance, one cannot return from one state to an earlier state; hence state accessibility here is not symmetric.

The question of using of temporal logic (TL) to software engineering was undertaken by Kröger [61,62,63,64]. The development of TL as applied to *CS* is due to Amir Pnueli. He was inspired by „Temporal Logic", a book written by Rescher and Urquhart [84].[6] „The Temporal Logic of Programs" [79], a paper by Pnueli,[7] is the classical source of TL for specification and verification of programs. This work is commonly seen as a crucial turning point in the progress of formal methods for the verification of concurrent and reactive systems. Amir Pnueli argues that temporal logic can be used as a formalism to reason about the behavior of computer programs and, in particular, of non-terminating concurrent systems.[8] In general, properties are

[5] Hoare's logic views a program as a transformation from an initial state to a final state. Thus it is not eligible to tackle problems of reactive or non-terminating systems, such as operating systems, where the computation does not bring to a final state.

[6] See [42, p. 222].

[7] Pnueli received the Turing Award in 1996:

> for seminal work introducing temporal logic into computing science and for outstanding contributions to program and system verification.

[8] A system is said to be concurrent when its behavior is the result of the interaction and evolution of multiple computing agents. The initial interest in concurrent systems was motivated by the speed improvements brought forth by multi-processor computers.

mostly describing correctness or safety of the system's operation. For Clarke [20, p. 1] works of Pnueli [79], Owicki and Lamport [78]:

> demonstrated convincingly that Temporal Logic was ideal for expressing concepts like mutual exclusion, absence of deadlock, and absence of starvation.

There is a difference between logician and computer scientists approach to systems of logics [14, p. 315]:

> Decidability and axiomatization are standard questions for logicians; but for practitioner, the important question is model-checking.

In opinion of Dijkstra:[9]

> The situation of programmer is similar to the situation of mathematician, who develops a theory and proves results. [. . .] One can never guarantee that a proof is correct, the best one can say, is: "I have not discovered any mistakes". [. . .] So extremely plausible, that the analogy may serve as a great source of inspiration. [. . .]
> Even under the assumption of flawlessly working machines we should ask ourselves the questions: "When an automatic computer produces results, why do we trust them, if we do so?" and after that; "What measures can we take to increase our confidence that the results produced are indeed the results intended?"

In another work [32, p. 6] Dijkstra says:

> Program testing can be used to show the presence of bugs, but never to show their absence.

Formulated in terms of Turing Machines, the verification problem was already considered by Turing [88]. He demonstrated that there is no general method of proving of correctness of any program.

Application of a computer system may cause not only material losses, e.g., in e-banking, but also may be dangerous for life, e.g., in health care, transportation, especially air and space flights.[10] Correctness of design is a very important factor of

[9] See Dijkstra E. W., *Programming Considered as a Human Activity*, `http://www.cs.utexas.edu/users/EWD/transcriptions/EWD01xx/EWD117.html`.

[10] A famous example: The Ariane-5 launch on June 4, 1996; it crashed 36 seconds after the launch due to a conversion of a 64-bit floating point into a 16-bit integer value. In 2008 it was announced that the Royal Navy was ahead of schedule for switching their nuclear submarines to a customized Microsoft Windows solution dubbed *Submarine Command System Next Generation*. In this case any error may have an unimaginable aftermath.

systems for preventing economical and human losses caused by minor errors. The reduction of errors in computer systems is one of the most important challenges of *CS* [64, p. V]. It has long been known that [36, p. 27]:

> computer software programs, computer hardware designs, and computer systems in general exhibit errors. Working programmers may devote more than half of their time on testing and debugging in order to increase reliability. A great deal of research effort has been and is devoted to developing improved testing methods. Testing successfully identifies many significant errors. Yet, serious errors still afflict many computer systems including systems that are safety critical, mission critical, or economically vital. The US National Institute of Standards and Technology has estimated that programming errors cost the US economy $60B annually[11].

Computer systems are more and more complicated. Verification of digital hardware designs has become one of the most expensive and time-consuming components of the current product development cycle. Empirical testing and simulation is expensive, not ultimately decisive and sometimes excluded for economical or ethical reasons. Formal methods are the most notable efforts to guarantee a correctness of system design and behaviors. Thus the formal specification and computer aided validation and verification are more and more indispensable. Formal methods have gained popularity in industry since the advent of the famous Intel Pentium *FDIV* bug in 1994, which caused *Intel* to recall faulty chips and take a loss of $475 million [28]. Digital computers are intended to be abstract discrete state machines and such machines and their software are naturally formalized in mathematical logic.

> Given the formal descriptions of such systems, it is then natural to reason about the systems by formal means. And with the aid of software to take care of the myriad details, the approach can be made practical. Indeed, given the cost of bugs and the complexity of modern hardware and software, these applications cry out for mechanical analysis by formal mathematical means. [59, p. 181–182]

Errors should already be detected at design stage. It is very important to specify the correctness property of system design and behavior, and an appropriate property must be specified to represent a correct requirement. It is estimated that 70% of design-time is spent to minimize the risk of errors [86], see [76]. Formal methods, model checkers as well theorem provers, are proposed as efficient, safe and less expensive tools [59,15]. According to Emerson [36, pp. 27–28]:

[11] See: National Institute of Standards and Technology, US Department of Commerce, “Software Errors Cost U.S. Economy $59.5 Billion Annually”, NIST News Release, June 28, 2002.

Given the incomplete coverage of testing, alternative approaches have been sought. The most promising approach depends on the fact that programs and more generally computer systems may be viewed as mathematical objects with behavior that is in principle well-determined. This makes it possible to specify using mathematical logic what constitutes the intended (correct) behavior. Then one can try to give a formal proof or otherwise establish that the program meets its specification. This line of study has been active for about four decades now. It is often referred to as *formal methods*.

2. Formal methods of verification

2.1 Formal methods

Formal methods include: formal specification, specification analysis and proof, transformational development, program verification. The principal benefits of formal methods are in reducing the number of faults in systems. Consequently, their main area of applicability is in critical systems engineering. There have been several successful projects where formal methods have been used in this area. The use of formal methods is most likely to be cost-effective because high system failure costs must be avoided. Nevertheless formal methods have not become mainstream software development techniques as was once predicted. Other software engineering techniques have been successful at increasing system quality. Hence the need for formal methods has been reduced. Market changes have made time-to-market rather than software with a low error count the key factor. Formal methods do not reduce time to market. Moreover, the scope of formal methods is limited. They are not well-suited to specifying and analyzing user interfaces and user interaction. Formal methods are still hard to scale up to large systems. Nevertheless as it is stressed by Edmund M. Clarke [56, p. ix], one of the prominent researcher in the field of formal methods in *CS*:

> Formal methods have finally come of age! Specification languages, theorem provers, and models checkers are beginning to be used routinely in industry.

The formal methods to be appropriate need to be properly adapted. Temporal logic and its language are of particular interest in the case of reactive[12], in particular concurrent systems. The language of TL is one that fulfills three important criteria. It:

[12] Systems can be divided into two categories: transformational programs (data intensive) and reactive systems (control intensive). The systems of the second type maintain an ongoing interaction with their environment (external and/or internal stimuli) and which ideally never terminate. Their specifications are typically expressed as constraints on their behavior over time.

- has the ability to express all sorts of specification (expressiveness);
- has reasonable complexity to evaluate the specified rules (complexity);
- due to its resemblance to natural language is easy to learn (pragmatics).

The knowledge of TL is indispensable in practice, tough, as it is remarked by Schnoebelen [87]:

> In today's curricula, thousands of programmers first learn about temporal logic in a course on model checking!

TL languages can be used to specification of widely spectrum of systems. Methods of TL can be applied to verification [72]. In the case of reactive systems TL is more useful than Floyd-Hoare logic that is better in the case of "input-output" programs. TL languages [64, p. 181]:

> provide general linguistic and deductive frameworks for *state systems* in the same manner as classical logics do for mathematical systems.

There are two main methods: proof-theoretical and model-theoretical [26].

2.2 Proof-theoretical approach

Already in the works of Turing the mathematical methods were applied to check correctness of programs [83]. By the end of sixties of last century Floyd [37], Hoare [48] and Naur [77] proposed axiomatic proving sequential programs with respect to their specification. Proof-theoretical method based on TL was proposed by Pnueli and Manna [72].

This method is used to prove a correctness of system through logical proving about system constraints or requirement for safe system behavior. Propositions specifying the system are joined as premisses to the thesis of deduction system of logic. Proofs can be "described" a variety of ways, e.g., by giving the inference steps, by specifying tactics or strategies to try, by stating the "landmark" subgoals or lemmas to establish, etc. Often, combinations of these styles are used within a single large proof project. Verification is positive if the proposition expressing the desired property is proved by using formal axioms and inference rules oriented towards sequential programs. Correctness of formal derivations could be "mechanically" checked, but finding a proof needs some experience and insight.

> But all proofs of commercially interesting theorems completed with mechanical theorem proving systems have one thing in common: they require a great deal of user expertise and effort. [59, pp. 182–183]

For example [59, p. 182]:

> The proof, constructed under the direction of this paper's authors and Tom Lynch, a member of the design team for the floating point unit, was completed 9 weeks after the effort commenced. About 1200 definitions and theorems were written by the authors and accepted, after appropriate proofs were completed by the *ACL*2[13] theorem prover.

At the time of its introduction in the early 1980's, a "manual" proof-theoretic approach was a prevailing paradigm for verification. Nowadays proofs are supported by semi-automatic means[14], *provers* and *proof checkers*. Interactive provers are used to partially automate the process of proving. Among the mechanical theorem proving systems used to prove commercially interesting theorems about hardware designs are *ACL*2[15], *Coq*[16], *HOL*[17], *HOL* Light[18], Isabelle[19], and *PVS*[20]. The proof assistant approach is a subject of research projects, e.g. *BRICKS* `http://www.bsik-bricks.nl/research_projects_afm4.shtml`.

The proof-theoretic framework is one-sided. It is possible only to prove that a proposition is a thesis. If we do not have a proof, we are entitled only to say that we could not find a proof, and nothing more. However, theorem proving can deal with an infinite state space, i.e., system with infinitely many configurations. Nevertheless this method is also indispensable in some intractable cases of finite state systems. Though today's model checkers are able to handle very large state spaces, eg. 10^{120} [[59, p. 183], [25]] but it does not mean that these states are explored explicitly. The above discussed theorem about *FDIV* (see p. 155) could be checked by running the microcode on about 10^{30} examples. Since in this case there are no reduction techniques, if it is assumed that one example could be checked in one femtosecond (10^{-15} seconds — the cycle time of a petahertz processor), the checking of the theorem will take more than 10^7 years [59, p. 183].

For Emerson [36, p. 28]:

> The need to encompass concurrent programs, and the desire to avoid the difficulties with manual deductive proofs, motivated the development of model

[13] See [71,12].

[14] Until the artificial intelligence problem is solved, human interaction will be important in theorem proving.

[15] See `http://www.cs.utexas.edu/~moore/acl2/`, [59].

[16] See `http://coq.inria.fr/`.

[17] See `http://www.cl.cam.ac.uk/research/hvg/HOL/`, [39].

[18] See `http://www.cl.cam.ac.uk/~jrh13/hol-light/`.

[19] See `http://www.cl.cam.ac.uk/research/hvg/Isabelle/`.

[20] See `http://pvs.csl.sri.com/`.

checking. In my experience, constructing proofs was sufficiently difficult that it did seem there ought to be an easier alternative.

2.3 Model-theoretical approach

Both the idea of automatic verification of concurrent programs based on model-theoretic approach and the term "model checking" were introduced by Clarke and Emerson in [21],[21] and independently the idea of model checking was conceived by Quille and Sifakis [82].[22] The idea was developed in works by Clarke, Emerson, Sistla and other [22,23,24,11,27].

Model checking is a verification technique that is preferred to theorem proving technique. This method, similarly as it is in the case of logical calculi, is more effective comparatively to proof-theoretic method. It is one of the most active research areas because its procedures are automatic and easy to understand.

According to Edmund M. Clarke [20, p. 1]:

> Model Checking did not arise in a historical vacuum. There was an important problem that needed to be solved, namely concurrent program verification.

In another place he continues:[23]

> Existing techniques for solving the problem were based on manual proof construction from program axioms. They did not scale to examples longer than a page and were extremely tedious to use. By 1981 the time was ripe for a new approach to the problem, and most of necessary ideas were already in place.

Model checking bridges the gap between theoretical computer science and hardware and software engineering. Model checking does not exclude the use of proof-theoretical methods, and conversely, the proof-theoretical methods do not exclude using of model checking. In practice one of theses methods is complementary to the other at least at the heuristic level. On the one hand, failed proofs can guide to the discovery of counterexamples. Any attempt of proving may be forego by looking for counterexamples. Counterexamples of consequences of a theorem can help to reformulate it. Examples may aid comprehension and invention of ideas and can be used as a basis for generalization being expressed by a theorem. The role of decision procedures is often essential in theorem proving. There has been considerable interest in

[21] See [36, p. 9].

[22] E. M. Clarke and E. A. Emerson interpreted concurrent system as finite Kripke structure/transition system and properties were expressed in CTL language. J.-P. Queille and J. Sifakis based on Petri nets and properties were expressed in language of branching time logic.

[23] See `http://events.berkeley.edu/index.php/calendar/sn/coe.html?event`.

developing theorem provers that integrate *SAT* solving algorithms. The efficient and flexible incorporating of decision procedures into theorem provers is very important for their successful use. There are several approaches for combining and augmenting of decision procedures. On the other hand, the combination of model checking with deductive methods allows the verification of a broad class of systems and, as it is in the case of eg. *STeP* [73], not restricted to finite-state systems. The question of combining proof-theoretical and model checking methods and the general problem of how to flexibly integrate decision procedures into heuristic theorem provers are subjects of many works [13].

In model checking the first task is to convert a system to a formal model accepted by a model checker. We model a system as a finite state machine. It is a model in the form of a Kripke structure[24] or labeled graph of state transitions — that has to accurately describe the behavior of the checked system. To do this formal languages defined by formal semantics must be used. To draw an abstract model many techniques are applied. Many methods are used to reduce states of a system. In practice, this process is not automated.

The second task is to specify properties that that must be satisfied by the real system. Mechanically assisted verification of properties of a complex system requires an accurate formal model of the system. The specification usually is given in some logical formalism. Generally, temporal logics are used to represent a temporal characteristic of systems.

We perform a model checker whether the system satisfies its properties as expressed by temporal logic formulas. The answer is positive only if all runs are models of the given temporal logic formula. The technique is based on the idea of exhaustive exploration of the reachable state space of a system. For this reason it can only be applied to systems with a finite state space, i.e., systems with finitely many configurations, and — for practical limitations (tractability) — with not too many states. The verification is completely automatic with the abstract model and properties. Thus it is possible to verify the correctness of very complicated and very large systems manual checking of which is almost not possible. We can verify a complex system as a hardware circuit or communication protocol automatically. The verification re-

[24] Kripke or relational semantics of modal logics has been conceived in fifties of the last century. This semantics was philosophically inspired nevertheless it has found application in *CS*. In *CS* Kripke structure is associated with a transition system. Because of the graphical nature of the state-space, it is sometimes referred to as the state graph associated with the system. Similarly as in modal logics this role may be played by Hintikka frames [8]. A Kripke frame consists of non-empty set and a binary relation defined on this set. In modal logics elements of the set are called possible worlds and the relation is understood as accessibility of one world from another. In the case of TL as applied in *CS* the Kripke semantics is based on computational time.

sults are correct and easy to analysis. However, it does need human assistance to analyze the result of model checking. If logic is complete with respect to the model and is decidable, then in the case of any proposition that specifies the behavior of the system the procedure of checking is finite. But if the model is too detailed the verification becomes intractable. A model checker verifies the model and generates verification results, "True" or counterexample if the result is "False". If the proposition is satisfied the system is verified. If the proposition is not valid the construction results in a counterexample — this is one of important advantages of model checking. The counterexample provides an information about an error (bug) in the system. The model checker can produce a counterexample for the checked property, and it can help the designer in tracking down where the error occurred.

The counterexample gives us a new precondition or a negative result in the following way: When we obtain a counterexample, we analyze it and as far as this trace could not occur in real system we add new preconditions to the formula. We may obtain a counterexample again which often results to many preconditions. In this case, analyzing the error trace may require a modification to the system and reapplication of the model checking process. The error can also result from incorrect modeling of the system or from an incorrect specification. The error trace can also be useful in identifying and fixing these two problems.

Model checking comes in two varieties depending on the way the proprieties are expressed. If theory of automata is employed the system as well as its specification are described by automaton. Questions concerning system and its specification are reduced to the question about the behavior of automaton. In other words, when we say "automata theoretic approach" we mean:

– specifying systems using automata
– reducing model checking to automata theory.

In the case of TL model checking the system is modeled as a finite-state automaton, while the specification is described in temporal language. A model checking algorithm is used to verify whether the automaton has the proper temporal-logical proprieties. In other words, if TL is applied [76, p. 2–3]:

> Model checking involves checking the truth of a set of specifications defined using a temporal logic. Generally, the temporal logic that is used is either CTL^* or one of its sublogics, CTL [...] [23] or LTL [...] [80].

Various model checkers are developed. They are applied to verification of large models, to real-time systems, probabilistic systems, etc. [50,66,24,10] — see [87].

Software is usually less structured than hardware and, especially in the case of concurrency, asynchronous. Thus the state space is bigger in the case of software than in hardware. Consequently, Model Checking has been used less frequently for software verification than for hardware verification [20, p. 18]. The limits of models checking are pushed by employing work-station clusters and *GRID*s, e.g. the *VeriGEM* project aims at using the storage and processing capacity of clusters of workstations on a nation-wide scale `www.bsik-bricks.nl/research_projects_afm6.shtml`. Despite being hampered by state explosion, since its beginning model checking has had a substantive impact on program verification efforts.

It is worth mentioning some of the applications of model checking elsewhere. These include understanding and analyzing legal contracts, which are after all prescriptions for behavior [31]; analyzing processes in living organisms for systems biology [43]; e-business processes such as accounting and workflow systems [91]. Model checking has also been employed for tasks in artificial intelligence such as planning [38]. Conversely, techniques from artificial intelligence related to SAT-based planning [60] are relevant to (bounded) model checking.

Let us repeat after Emerson some interesting remarks concerning model checking [36, p. 42]:

> Edsger W. Dijkstra commented to me that it was an "acceptable crutch" if one was going to do after-the-fact verification. When I had the pleasure of meeting Saul Kripke and explaining model checking over Kripke structures to him, he commented that he never thought of that. Daniel Jackson has remarked that model checking has "saved the reputation" of formal methods.

3. Model checkers

By a model checker we mean a procedure which checks if a transition system system is a model for a formula expressing a certain property of this system [23].

There is a wide variety of model checkers available, with a number of different capabilities suited to different kinds of problems. Some of these are academic tools, others are industrial internal tools, and some are for sale by *CAD* vendors. The variety is of great benefit to practitioners. They have to know which tools are available and which tools to chose for a particular problem. Today, software, hardware and *CAD* companies employ several kinds of model checkers. In software, *Bell Labs*, *JPL*, and *Microsoft*, government agencies such as *NASA* in USA, in hardware and *CAD*, *IBM*, *Intel* (to name a few) have had tremendous success using model checking for verifying switch software, flight control software, and device drivers.

Some programs are grouped as it is in the case of MODEL-CHECKING KIT `http://www.fmi.uni-stuttgart.de/szs/tools/mckit/overview.shtml`. This is a collection of programs which allow to model a finite-state system using a variety of modeling languages, and verify it using a variety of checkers, including deadlock-checkers, reachability-checkers, and model-checkers for the temporal logics CTL and LTL. The most interesting feature of the Kit is that:

> Independently of the description language chosen by the user, (almost) all checkers can be applied to the same model.

The counterexamples produced by the checker are presented to the user in terms of the description language used to model the system.

The Kit is an open system: new description languages and checkers can be added to it.

The description languages and the checkers have been provided by research groups at the *Carnegie-Mellon University*, the *University of Newcastle upon Tyne*, *Helsinki University of Technology*, *Bell Labs*, the *Brandenburg Technical University at Cottbus*, the *Technical University of Munich*, the *University of Stuttgart*, and the *Humboldt-Universität zu Berlin*.

Problems of techniques and tools of verification of *ICT* systems are subjects of research projects. E.g., in the scheme of *BRICKS* `http://www.bsik-bricks.nl/index.shtml` under theme *Algorithms and Formal Methods* there are developed

– Advancing the Real Use of Proof Assistants
– Infinite Objects: Computation, Modeling and Reasoning
– A Verification Grid for Enhanced Model Checking
– Modeling and Analysis of QoS for Component-Based Designs
– A Common Framework for the Analysis of Reactive and Timed Systems

Many of the research problems originating from industrial parties.

Below we give a few examples of model checkers. Usually they description will be taken from they website home pages.

Two of the most popular on-the-fly, explicit-state-based model checkers are SPIN (**S**imple **P**romela **IN**terpreter) and MURϕ or MURPHI [35,34].

SPIN is:

> a popular open-source software tool, used by thousands of people worldwide, that can be used for the formal verification of distributed software systems. The tool was developed at *Bell Labs* in the original UNIX group of the Computing Sciences Research Center, starting in 1980. The software has been

available freely since 1991, and continues to evolve to keep pace with new developments in the field. In April 2002 the tool was awarded the prestigious *System Software Award* for 2001 by the *ACM*. `http://spinroot.com/spin/whatispin.html`

SPIN continues to evolve to keep pace with new developments in the field. The DSPIN tool [57] is an extension of SPIN, which has been designed for modeling and verifying object-oriented software (JAVA programs, in particular).

Murφ is a system description high-level language and model checker developed to formally evaluate behavioral requirements for finite-state asynchronous concurrent systems [35,34], `http://sprout.stanford.edu/dill/murphi.html`. *Murφ* is developed by a research group at the University of Utah `http://www.cs.utah.edu/formal_verification/`.

SMV `http://www.cs.cmu.edu/~modelcheck/smv.html` (**S**ymbolic **m**odel **v**erifier) is a model checker that accepts both the temporal logics *LTL* and *CTL*. It is the first and the most successful *OBDD*-based symbolic model checker [75]. SMV has been developed by The Model Checking Group that is a part of Specification and Verification Center, Carnegie Mellon University `http://www-2.cs.cmu.edu/~modelcheck/index.html`.

CADENCE SMV `http://www.kenmcmil.com/smv.html` is a symbolic model checking tool released by Cadence Berkeley Labs. CADENCE SMV is provided for formal verification of temporal logic properties of finite state systems, such as computer hardware designs. It is an extension of SMV. It has a more expressive mode description language, and also supports synthesizable VERILOG as a modeling language.

NUSMV `http://nusmv.irst.itc.it`, `http://nusmv.fbk.eu` is an updated version of SMV [18,17]. The additional features contained in NUSMV include a textual interaction shell and graphical interface, extended model partitioning techniques, and facilities for *LTL* model checking. NUSMV [19] has been developed as a joint project between Formal Methods group in the Automated Reasoning System division at Istituto Trentino di Cultura, Istituto per la Ricerca Scientifica e Tecnologica in Trento, Italy), the Model Checking group at Carnegie Mellon University, the Mechanized Reasoning Group at the University of Genoa and the Mechanized Reasoning Group at the University of Trento.

NUSMV 2 is open source software. It combines BDD-based model checking with SAT-based model checking. It has been designed as an open architecture for model checking. NUSMV 2 exploits the CUDD library developed by Fabio Somenzi at Colorado University and SAT-based model checking component that includes an

RBC-based Bounded Model Checker, connected to the SIM SAT library developed by the University of Genova. It is aimed at reliable verification of industrially sized designs, for use as a back-end for other verification tools and as a research tool for formal verification techniques.

An enhanced version of SMV, RULEBASE www.haifa.ibm.com/projects/verification/RB_Homepage/ [7] is an industry-oriented tool for the verification of hardware designs, developed by the IBM Haifa Research Laboratory. In an effort to make the specification of CTL properties easier for the non-expert, RULEBASE supports its own language, Sugar. In addition, RULEBASE supports standard hardware description languages such as VHDL and VERILOG. RULEBASE is especially applicable for verifying the control logic of large hardware designs.

VEREOFY http://www.vereofy.de/ was written at Technische Universität Dresden. It is developed in the context of the EU project CREDO. VEREOFY is a formal verification tool of checking of component-based systems for operational correctness.

Model checking tools were initially developed to reason about the logical correctness of discrete state systems, but have since been extended to deal with real-time and limited forms of hybrid systems. Real-time systems are systems that must perform a task within strict time deadlines. Embedded controllers, circuits and communication protocols are examples of such time-dependent systems. The hybrid model checker HYTECH [44] is used to analyze dynamical systems whose behavior exhibits both discrete and continuous change. HYTECH automatically computes the conditions on the parameters under which the system satisfies its safety and timing requirements.

The most widely used dense real-time model checker (in which time is viewed as increasing continuously) is UPPAAL www.uppaal.com/ [70]. Models are expressed as timed automata [2] and properties defined in UPPAAL logic, a subset of Timed Computational Tree Logic ($TCTL$) [1]. UPPAAL is an integrated tool environment for modeling, validation and verification of real-time systems modeled as networks of timed automata, extended with data types (bounded integers, arrays, etc.). The tool is developed in collaboration between the Department of Information Technology at Uppsala University, Sweden and the Department of Computer Science at Aalborg University in Denmark.

Another real-time model checker is KRONOS http://www-verimag.imag.fr/TEMPORISE/kronos/ [92]. KRONOS is developed at VERIMAG, a leading research center in embedded systems in France. KRONOS checks whether a real-time system modeled by a timed automaton satisfies a timing property specified by a formula of the **T**imed **C**omputational **T**ree **L**ogic $TCTL$, a timed extension of CTL.

Model checking requires the manual construction of a model, via a modeling language, which is then converted to a Kripke structure or an automaton for model checking. Model checking starts with translation to model checker language. In model checking considerable gains can be made by finding ways to extract models directly from program source code. There have been several promising attempts to do so.

VeriSoft `http://cm.bell-labs.com/who/god/verisoft/` is the first model checker that could handle programs directly.

The first version of Blast (**B**erkeley **L**azy **A**bstraction **S**oftware verification **T**ool) `http://mtc.epfl.ch/software-tools/blast/`, `http://www.sosy-lab.org/~dbeyer/blast_doc/blast001.html`, `www.sosy-lab.org/~dbeyer/blast_doc/blast.pdf` [45] was developed for checking safety properties in C programs at University of California, Berkeley. The Blast project is supported by the National Science Foundation. Blast is a popular software model checker for revealing errors in `Linux` kernel code. Blast is relatively independent of the underlying machine and operating system. It is free software, released under the Modified *BSD* license `http://www.oss-watch.ac.uk/resources/modbsd.xml`. Blast is based on similar concepts as Slam `http://research.microsoft.com/en-us/projects/slam/`. Blast and Slam are relatively new. Slam was developed by Microsoft Research around 2000, i.e., earlier than Blast, which was developed around 2002. Both the checkers have many characteristics in common. One key difference between Slam and Blast is the use of lazy abstraction in Blast.

Slam has been customized for the `Windows` product `StaticDriverVerifier`, Sdv, a tool in the `WindowsDriverDevelopmentKit`.

Slam and Blast differ from other model checking tools in many ways. First of all, the traditional approach to model-checking (followed by Spin and Kronos) has been to first create a model of a system, and once the model has been verified, move on to the actual implementation. Slam and Blast fall in the category of the "modern" approach in model checking. The user has already completed the implementation and wishes to verify the software. The objective then is to create a model from the existing program and apply model checking principles, such that the original program is verified.

The FeaVer (**Fea**ture **Ver**ification system) `http://cm.bell-labs.com/cm/cs/what/feaver/` tool grew out of an attempt to come up with a thorough method to check the call processing software for a commercial switching product, called the PathStar® access server [54,51]. It allows models to be extracted mechanically from the source of software applications, and checked using SPIN. SPIN allows C code to be embedded directly within a PROMELA specification [53,52].

The Time Rover `http://www.time-rover.com/` is a specification based verification tool for applications written in C, C++, JAVA, VERILOG and VHDL. The tool combines formal specification, using LTL and MTL, with conventional simulation/execution based testing. The Temporal Rover is tailored for the verification of complex protocols and reactive systems where behavior is time dependent. The methodology and technology are based on the Unified Modeling Language (UML) and are currently in active use by NASA and the national Missile Defense development team.

Since Pnueli introduced temporal logic to computer science, the logic has been extended in various ways to include probability. Probabilistic techniques have proved successful in the specification and verification of systems that exhibit uncertainty. Early works in this field were focusing on the verification of qualitative properties. These included work of [30] which considered models of two types, **D**iscrete-**T**ime **M**arkov **C**hains (DTMCS) and **M**arkov **D**ecision **P**rocesses (MDPS).

Tools concerning model checking probabilistic systems such as PRISM (**PR**obabilistic **S**ymbolic **M**odel Checker) `http://www.cs.bham.ac.uk/~dxp/prism/`, [67,69,68] have been developed and applied to several real-world case studies. Other tools include ETMCC [46], CASPA [65] and MRMC (**M**arkov **R**eward **M**odel **C**hecker) [58].

ETMCC [90] requirements against action-labeled continuous time Markov chains. Probabilistic Model Checker ETMCC (**E**rlangen-**T**wente **M**arkov **C**hain **C**hecker) [46] is developed jointly by the Stochastic Modeling and Verification group at the University of Erlangen-Nürnberg, Germany, and the Formal Methods group at the University of Twente, the Netherlands. ETMCC is the first implementation of a model checker for **D**iscrete-**T**ime **M**arkov **C**hain**s** (DTMS) and **C**ontinuous-**T**ime **M**arkov **C**hains (CTMCS). It uses numerical methods to model check $PCTL$ [41] and **C**ontinuous **S**tochastic **L**ogic (*CSL*)[25] formulas respectively for DTMCS and CTMCS.

Markov Reward Model Checker **M**arkov **R**eward **M**odel **C**hecker (MRMC) `http://www.mrmc-tool.org/trac/` has been developed by the Formal Methods & Tools group at the University of Twente, The Netherlands and the Software Modeling and Verification group at RWTH Aachen University, Germany under the guidance of Joost-Pieter Katoen [5, Ch. 10 Probabilistic systems]. MRMC is a successor of ETMCC, which is a prototype implementation of a model checker for continuous-time Markov chains.

[25] A branching-time temporal logic a´ la CTL with state and path formulas [4,6,3].

PRISM stands for Probabilistic Symbolic Model Checker `http://www.prismmodelchecker.org/`. It is the internationally leading probabilistic model checker being implemented at the University of Birmingham [67,69,68], `http://www.cs.bham.ac.uk/~dxp/prism/`. First public release: September 2001.

There are three types of probabilistic models that PRISM can support directly: **D**iscrete-**T**ime **M**arkov **C**hain**s**, Markov decision processes and **C**ontinuous-**T**ime **M**arkov **C**hain**s**.

PRISM [69,85] allows time to be considered as increasing either in discrete steps or continuously. Models are expressed in PRISM own modeling language and converted to a variant of a Markov chain (either discrete- or continuous-time). Properties are written in terms of *PCTL* or *CSL*, respectively. Models can also be expressed using PEPA (**P**erformance **E**valuation **P**rocess **A**lgebra) [47] and converted to PRISM. PRISM is free and open source, released under the *GNU* General Public License (*GPL*), available freely for research an teaching.

References

[1] Alur R., Courcoubetis C., Dill D. L. (1990): Model-checking for real-time systems, *in* 'Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science', IEEE Computer Society Press, Philadelphia, PA, pp. 414–425.

[2] Alur R., Dill D. (1993): A theory of timed automata', *Inf. Comput.* **194**, 2–34.

[3] Aziz A., Sanwal K., Singhal V., Brayton R. (2000): Model checking continuous time Markov chains, *ACM Trans. Computational Logic* **1**(1), 162–170.

[4] Aziz A., Sanwal K., Singhal V., Brayton R. K. (1996): Verifying continuous time Markov chains, *in* R. Alur, T. A. Henzinger, eds, 'Eighth International Conference on Computer Aided Verification CAV 1996', Vol. 1102 of *Lecture Notes in Computer Science*, Springer Verlag, New Brunswick, NJ, USA, pp. 269–276.

[5] Baier C., Katoen J. P. (2008): *Principles of Model Checking*, The MIT Press. Foreword by Kim Guldstrand Larsen.

[6] Baier C., Katoen J.-P., Hermanns H. (1999): Approximate symbolic model checking of continuous-time Markov chains, *in* 'International Conference on Concurrency Theory', pp. 146—-161.

[7] Beer, I., Ben-David, S., Eisner, C., Landver, A. (1996): Rulebase: An industry-oriented formal verification tool, *in* 'Proceedings of the 33rd Conference on Design Automation (DAC'96)', ACM Press, Las Vegas, NV, pp. 655—660.

[8] Ben-Ari, M., Manna, Z., Pnueli, A. (1981): The temporal logic of branching time, *in* 'Proc. 8th ACM Symposium on Principles of Programming Languages', ACM Press, New York, pp. 164–176. Por. [9].

[9] Ben-Ari, M., Manna, Z., Pnueli, A. (1983): 'The temporal logic of branching time', *Acta Informatica* **20**, 207–226. Por. [8].

[10] Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., Schnoebelen, P. (2001): *Systems and Software Verification. Model-Checking Techniques and Tools*, Springer.

[11] Bidoit, B. M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L. , Schnoebelen, P. (2001): *Systems and Software Verification: Model-checking Techniques and Tools*, Springer.

[12] Boyer, R. S., Moore, J. S. (1979): *A Computational Logic*, Academic Press, New York.

[13] Boyer, R. S., Moore, J. S. (1988): 'Integrating decision procedures into heuristic theorem provers: A case study of linear arithmetic', *Machine Intelligence* **11**, 83–124.

[14] Bradfield, J. C., Stirling, C. (2001): Modal logics and μ-calculi: An introduction, *in* J. A. Bergstra, A. Ponse, S. A. Smolka, eds, 'Handbook of Process Algebra', Elsevier Science, chapter 4, pp. 293–330.

[15] Brock, B., Hunt, W. (1997): Formally specifying and mechanically verifying programs for the motorola complex arithmetic processor dsp, *in* 'Proceedings of the IEEE International Conference on Computer Design (ICCD'97)', pp. 31—-36.

[16] Cengarle, M. V., Haeberer, A. M. (2000): Towards an epistemology-based methodology for verification and validation testing, Technical report 0001, Ludwig-Maximilian's Universität, Institut für Informatik, München, Oettingenstr. 67. 71 pages.

[17] Cimatti, A., Clarke, E., Giunchig lia, E., Giunchig lia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A. (2002): NUSMV2: A new opensource tool for symbolic model checking, *in* E. Brinksma, K. Larsen, eds, 'Proceedings of the 14th International Conference on Computer-Aided Verification (CAV 2002)', Vol. 2404 of *Lecture Notes in Computer Science*, Springer-Verlag, Copenhagen, Denmark, pp. 359—-364.

[18] Cimatti, A., Clarke, E., Giunchig lia, F., Roveri, M. (1999): NUSMV2: A new symbolic model verifier, *in* N. Halbwachs, D. Peled, eds, 'Proceedings of the 11th International Conference on Computer-Aided Verification (CAV '99)', Vol. 1633 of *Lecture Notes in Computer Science*, Springer-Verlag, Trento, Italy, pp. 495—499.

[19] Cimatti, A., Clarke, E. M., Giunchig lia, F., Roveri, M. (2000): 'NUSMV: A new symbolic model checker', *International Journal on Software Tools for Technology Transfer* **2**(4), 410–425.

[20] Clarke, E. M. (2008): The birth of model checking, *in* DBLP:conf/spin/5000, pp. 1–26.

[21] Clarke, E. M., E., E. A. (1982): Design and synthesis of synchronization skeletons using branching-time temporal logic, *in* 'Logic of Programs, Workshop', Vol. 131 of *Lecture Notes in Computer Science*, Springer-Verlag, London, UK, pp. 52—-71.

[22] Clarke, E. M., Emerson, E. A., Sistla, A. P. (1983): Automatic verification of finite state concurrent systems using temporal logic specifications: A practical approach, *in* 'Conference Record of the Tenth Annual ACM Symposium on Principles of Programming Languages', Austin, Texas, pp. 117–126.

[23] Clarke, E. M., Emerson, E. A., Sistla, A. P. (1986): 'Automatic verification of finite-state concurrent systems using temporal logic specifications', *ACM Transactions on Programming Languages and Systems* **8**(2), 244–263.

[24] Clarke, E. M., Grumberg, J. O., Peled, D. A. (1999): *Model Checking*, The MIT Press.

[25] Clarke, E. M., Grumberg, O., Jha, S., Lu, Y., Veith, H. (2001): Progress on the state explosion problem in model checking, *in* 'Informatics — 10 Years Back. 10 Years Ahead.', Vol. 2000 of *Lecture Notes in Computer Science*, Springer-Verlag, London, UK, pp. 176–194.

[26] Clarke, E. M., Wing, J. M., Alur, R., Cleaveland, R., Dill, D., Emerson, A., Garland, S., German, S., Guttag, J., Hall, A., Henzinger, T., Holzmann, G., Jones, C., Kurshan, R., Leveson, N., McMillan, K., Moore, J., Peled, D., Pnueli, A., Rushby, J., Shankar, N., Sifakis, J., Sistla, P., Steffen, B., Wolper, P., Woodcock, J., Zave, P. (1996): 'Formal methods: state of the art and future directions', *ACM Computing Surveys* **28**(4), 626–643.

[27] Clarke, E., Wing, J. M. (1996): 'Formal methods: State-of-the-art and future directions', *ACM Comput. Surv.* **28**(4), 626—-643. Report by the Working Group on Formal Methods for the ACM Workshop on Strategic Directions in Computing Research.

[28] Coe, T., Mathisen, T., Moler, C., Pratt, V. (1995): 'Computational aspects of the pentium affair', *IEEE Comput. Sci. Eng.* **2**(1), 18–31.

[29] Connelly, R., Gousie, M. B., Hadimioglu, H., Ivanov, L., Hoffman, M. (2004): 'The role of digital logic in the computer science curriculum', *Journal of Computing Sciences in Colleges* **19**, 5–8.

[30] Courcoubetis, C., M. Yannakakis, M. (1988): Verifying temporal properties of finite state probabilistic programs, *in* 'Proc. 29th Annual Symposium on Foundations of Computer Science (FOCS'88)', IEEE Computer Society Press, pp. 338—-345.

[31] Daskalopulu, A. (2000): Model checking contractual protocols, *in* J. Breuker, R. Leenes, R. Winkels, eds, 'Legal Knowledge and Information Systems', JURIX 2000: The 13th Annual Conference, IOS Press, Amsterdam, pp. 35–47.

[32] Dijkstra, E. W. (1968): Notes on structured programming, *in* E. W. D. O.-J. Dahl, C. A. R. Hoare, eds, 'Structured Programming', Academic Press, London, pp. 1–82.

[33] Dijkstra, E. W. (1989): In reply to comments. EWD1058.

[34] Dill, D. L. (1996): The murϕ verification system, *in* R. Alur, T. Henzinger, eds, 'Proceedings of the 8th International Conference on Computer Aided Verification (CAV '96)', Vol. 1102 of *Lecture Notes in Computer Science*, Springer-Verlag, New Brunswick, NJ, pp. 390—-393.

[35] Dill, D. L., Drexler, A. L., Hu, A. J., Yang, C. H. (1992): Protocol verification as a hardware design aid, *in* 'Proceedings of the 1992 IEEE International Conference on Computer Design: VLSI in Computer and Processors (ICCD'92)', IEEE Computer Society, Cambridge, MA, pp. 522–525.

[36] Emerson, E. A. (2008): The beginning of model checking: A personal perspective, *in* DBLP:conf/spin/5000, pp. 27–45.

[37] Floyd, R. W. (1967): Assigning meanings to programs, *in* J. T. Schwartz, ed., 'Mathematical Aspects of Computer Science. Proceedings of Symposia in Applied Mathematics', Vol. 19, American Mathematical Society, Providence, pp. 19–32.

[38] Giunchig lia, F., Traverso, P. (1999): Planning as model checking, *in* 'Proceedings of the Fifth European Workshop on Planning, (ECP'99)', Springer, pp. 1–20.

[39] Gordon, M., Melham, T. (1993): *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*, Cambridge University Press.

[40] Grumberg, O., Veith, H., eds (2008): *25 Years of Model Checking - History, Achievements, Perspectives*, Vol. 5000 of *Lecture Notes in Computer Science*, Springer.

[41] Hansson, H., Jonsson, B. (1994): 'A logic for reasoning about time and reliability', *Formal Aspects of Computing* **6**, 512–535.

[42] Hasle, P. F. V., Øhrstrøm, P. (2004): Foundations of temporal logic. The WWW-site for Arthur Prior, `http://www.kommunikation.aau.dk/prior/index2.htm`.

[43] Heath, J., Kwiatowska, M., Norman, G., Parker, D., Tymchysyn, O. (2006): Probabalistic model checking of complex biological pathways, *in* C. Priami, ed., 'Proc. Comp. Methods in Systems Biology, (CSMB'06)', Vol. 4210 of *Lecture Notes in Bioinformatics*, Springer, pp. 32–47.

[44] Henzinger, T., Ho, P., Wong-Toi, H. (1997): 'A model checker for hybrid systems', *Int. J. Softw. Tools Technol. Transfer* **1**(1/2), 110–122.

[45] Henzinger, T., Jhala, R., Majumdar, R., Sutre, G. (2003): Software verification with BLAST, *in* T. Ball, S. Rajamani, eds, 'Model Checking Software: Proceedings of the 10th International SPIN Workshop (SPIN 2003)', Vol. 2648 of *Lecture Notes in Computer Science*, Springer-Verlag, Portland, OR, pp. 235–239.

[46] Hermanns, H., Katoen, J.-P., Meyer-Kayser, J., Siegle, M. (2000): A Markov chain model checker, *in* 'Tools and Algorithms for Construction and Analysis of Systems', pp. 347—-362.

[47] Hillston, J. (1996): *A Compositional Approach to Performance Modeling*, Distinguished Dissertations in Computer Science, Cambridge University Press, Cambridge, UK.

[48] Hoare, C. A. R. (1969): 'An axiomatic basis for computer programming', *Communications of the ACM* **12**(10), 576–580,583. Również w: [49, 45–58].

[49] Hoare, C. A. R., Jones, C. B. (1989): *Essays in Computing Science*, Prentice Hall.

[50] Holzmann, G. (1991): *Design and validation of computer protocols*, Prentice Hall, New Jersey.

[51] Holzmann, G. J. (2002): Software analysis and model checking, *in* 'CAV', pp. 1–16.

[52] Holzmann, G. J., Smith, M. H. (2002): FEAVER 1.0 user guide, Technical report, Bell Labs. 64 pgs.

[53] Holzmann, G., Smith, M. (1999): A practical method for the verification of event-driven software, *in* 'Proceedings of the 21st International Conference on Software engineering (ICSE '99), Los Angeles, CA', ACM Press, New York, pp. 597–607.

[54] Holzmann, G., Smith, M. (1999): Software model checking. Extracting verification models from source code, *in* J. W. et al., ed., 'Proceedings of the Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols and Protocol Specification, Testing and Verification (FORTE/PSTV '99)', Vol. 156, International Federation for Information Processing, Kluwer, Beijing, China, pp. 481–497.

[55] Hughes, G. E., Cresswell, M. J. (1968): *An Introduction to Modal Logic*, Methuen and Co., London.

[56] Huth, M. R. A., D., R. M. (2000): *Logic in Computer Science: Modelling and Reasoning about Systems*, Cambridge University Press.

[57] Iosif, R., Sisto, R. (1999): dspin: A dynamic extension of spin, *in* D. D. et al., ed., 'Proceedings of the 5th and 6th International SPIN Workshops', Vol.

1680 of *Lecture Notes in Computer Science*, Springer-Verlag, Trento, Italy and Toulouse, France, pp. 20–33.

[58] Katoen, J.-P., Khattri, M., Zapreev, I. S. (2005): A Markov reward model checker, *in* 'Quantitative Evaluation of Systems (QEST)', pp. 243—-244.

[59] Kaufmann, M., Moore, J. S. (2004): 'Some key research problems in automated theorem proving for hardware and software verification', *Rev. R. Acad. Cien. Serie A. Mat.* **98**(1), 181—-196.

[60] Kautz, H., Selman, B. (1992): Planning as satisfiability, *in* 'ECAI '92: Proceedings of the 10th European conference on Artificial intelligence', John Wiley & Sons, Inc., New York, NY, USA, pp. 359–363.

[61] Kröger, F. (1977): 'A logic of algorithmic reasoning', *Acta Informatica* **8**(3), 243–266.

[62] Kröger, F. (1987): *Temporal Logic of Programs*, Springer-Verlag New York, Inc., New York, NY, USA.

[63] Kröger, F., Merz, S. (1991): 'Temporal logic and recursion', *Fundam. Inform.* **14**(2), 261–281.

[64] Kröger, F., Merz, S. (2008): *Temporal Logic and State Systems*, Springer.

[65] Kuntz, M., Siegle, M., Werner, E. (2004): *Symbolic performance and dependability evaluation with the tool CASPA*.

[66] Kurshan, R. (1995): *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach*, Princeton Series in Computer Science, Princeton University Press, Princeton, NJ.

[67] Kwiatkowska, M., Norman, G., Parker, D. (2001): *PRISM*: Probabilistic symbolic model checker, *in* P. Kemper, ed., 'Proc. Tools Session of Aachen 2001', International Multiconference on Measurement, Modelling and Evaluation of Computer-Communication Systems, Dortmund, pp. 7–12. Available as Technical Report 760/2001, University of Dortmund.

[68] Kwiatkowska, M., Norman, G., Parker, D. (2002): Probabilistic symbolic model checking with *PRISM*: A hybrid approach, *in* J.-P. Katoen, P. Stevens, eds, 'Proc. 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)', Vol. 2280 of *Lecture Notes in Computer Science*, Springer, pp. 56–66.

[69] Kwiatkowska, M., Norman, G., Parker, D. (2002): Probabilistic symbolic model checking with PRISM, *in* J. Katoen, P. Stevens, eds, 'Proceedings of the 8th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS 2002)', Vol. 2280 of *Lecture Notes in Computer Science*, Springer-Verlag, Grenoble, France, pp. 52–66. Held as part of the Joint European Conference on Theory and Practice of Software (ETAPS 2002).

[70] Larson, K., Pettersson, P., Yi, W. (1997): 'Uppaal in a nutshell', *Int. J. Softw. Tools. Technol. Transfer* **1**(1/2), 134–152.

[71] M. Kaufmann, M., Manolios, P., Moore, J. S. (2000): *Computer-Aided Reasoning: An Approach*, Kluwer Academic Press, Boston.

[72] Manna, Z., A. Pnueli, A. (1992, 1995): *The Temporal Logic of Reactive and Concurrent Systems*, Vol. 1: Specification, 2: Safety, Springer-Verlag, New York.

[73] Manna, Z., Bjørner, N., Browne, A., Chang, E., Alfaro, L. D., Devarajan, H., Kapur, A., Lee, J., Sipma, H. (1994): STEP: The stanford temporal prover, Technical report, Computer Science Department, Stanford University Stanford, CA.

[74] Manna, Z., Waldinger, R. (1985): *The Logical Basis for Computer Programming*, Addison-Wesley.

[75] McMillan, K. L. (1993): *Symbolic Model Checking: An approach to the State Explosion Problem*, Kluwer Academic, Hingham, MA.

[76] Miller, A., Donaldson, A., Calder, M. (2006): 'Symmetry in temporal logic model checking', *ACM Computing Surveys* **38**(3).

[77] Naur, P. (1966): 'Proof of algorithms by general snapshots', *BIT* **6**(4), 310–316.

[78] Owicki, S. S., Lamport, L. (1982): 'Proving liveness properties of concurrent programs', *ACM Trans. Program. Lang. Syst.* **4**(3), 455–495.

[79] Pnueli, A. (1977): The temporal logic of programs, *in* 'Proceedings of the 18th IEEE-CS Symposium on Foundation of Computer Science (FOCS-77)', IEEE Computer Society Press, pp. 46–57.

[80] Pnueli, A. (1981): 'The temporal semantics of concurrent programs', *Theoretical Comput. Sci.* **13**, 45–60.

[81] Pratt, V. R. (1980): 'Applications of modal logic to programming', *Studia Logica* **9**, 257–274.

[82] Queille, J.-P., Sifakis, J. (1982): Specification and verification of concurrent systems in CESAR, *in* 'Proceedings 5th International Symposium on Programming', Vol. 137 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 337–351.

[83] Randell, B. (1973): *The Origin of Digital Computers*, Springer Verlag.

[84] Rescher, N., Urquhart, A. (1971): *Temporal Logic*, Springer, Wien, New York.

[85] Rutten, J., Kwiatkowska, M., Norman, G., Parker, D. (2004): *Mathematical Techniques for Analysing Concurrent and Probabilisitic Systems*, Vol. 23 of *American Mathematical Society, CRM Monograph Series*, Centre de Recherches Mathématiques, Université de Montréal.

[86] Schneider, K. (2003): *Verification of Reactive Systems. Formal Methods and Algorithms*, Texts in Theoretical Computer Science (EATCS Series), Springer-Verlag.

[87] Schnoebelen, P. (2002): 'The complexity of temporal logic model checking', *Advances in Modal Logic* **4**, 1–44.

[88] Turing, A. M. (1936–37): 'On computable numbers, with an application to the Entscheidungsproblem', *Proceedings of the London Mathematical Society* **42**(Series 2), 230–265. Received May 25, 1936; Appendix added August 28; read November 12, 1936; corrections Ibid. vol. 43(1937), pp. 544–546. Turing's paper appeared in Part 2 of vol. 42 which was issued in December 1936 (Reprint in: [89]; 151–154). Online version: `http://www.abelard.org/turpap2/tp2-ie.asp`.

[89] Turing, A. M. (1965): On computable numbers, with an application to the Entscheidungsproblem, *in* M. Davis, ed., 'The Undecidable', Raven Press, Hewlett, NY, pp. 116–151.

[90] Vaandrager F. W.and De Nicola, R. (1990): Actions versus state based logics for transition systems, *in* 'Proc. Ecole de Printemps on Semantics of Concurrency', Vol. 469 of *Lecture Notes in Computer Science*, Springer, pp. 407—-419.

[91] Wang, W., Hidvegi, Z., Bailey, A., Whinston, A. (2000): 'E-process design and assurance using model checking', *IEEE Computer* **33**(10), 48–53.

[92] Yovine, S. (1997): 'Kronos: A verification tool for real-time systems', *Int. J. Softw. Tools Technol. Transfer* **1**(1/2), 123–133.

LOGIKA I FORMALNA WERYFIKACJA SYSTEMÓW KOMPUTEROWYCH. KILKA UWAG.

Streszczenie Do specyfikacji i weryfikacji zarówno sprzętu jak i programów stosowane są różne logiki. Główną wadą metody teorio-dowodowej weryfikacji jest problem znalezienia dowodu. Zastosowanie tej metody zakłada aksjomatyzację logiki. Własności systemu mogą być sprawdzane za pomocą jego modelu. Model jest zbudowany w języku specyfikacji i sprawdzany automatycznie. Zastosowanie sprawdzania za pomocą modelu zakłada rozstrzygalność zadania. Istnieje wielka różnorodność programów (model checker) do sprawdzania własności za pomocą modeli.

Słowa kluczowe: Logika, Weryfikacja, Metoda teorio-dowodowa, Sprawdzanie za pomocą modelu

Praca wspierana przez grant MNiSW nr 3 T11F 01130.