# On Application of Machine Learning for Development of Adaptive Sorting Programs in Algebra of Algorithms

Olena Yatsenko

Institute of Software Systems of National Academy of Sciences of Ukraine,
Glushkov prosp. 40, 03187 Kyiv, Ukraine
oayat@ukr.net

**Abstract.** The experiment aimed at development of adaptive sorting program on the basis of usage of algorithm selection method, machine learning system and algebra-algorithmic approach is conducted. Machine learning facilities allow to automatize constructing of adaptive algorithm on the basis of analysis of experimental data, related to execution of initial algorithms. Designing of algorithms is based on usage of systems of algorithmic algebras.

**Keywords:** algebras of algorithms, algorithm selection, data mining, decision tree, formalized designing of programs, machine learning, sorting.

## 1 Introduction

In recent years the necessity for developing applications, which are capable to analyze the information arriving during the algorithm execution, carry out forecasting and adapt to new circumstance, increases more and more. Development of such programs is associated with accumulation and analysis of a considerable quantity of experimental data (input, intermediate and resulting). Thus there is a problem of automation of extraction of helpful information from large amount of data. This problem is being solved within the bounds of data mining technology [18], which is one of actively developing areas of information technologies. Data mining is intended for extracting useful knowledge from large data sets by combining methods from statistics and artificial intelligence. There are works in several subject domains applying data mining, machine learning and algorithm selection methods for developing adaptive programs: time-series forecasting [16]; constraint satisfaction problems [11]; optimization [14]; sorting [7, 9].

The novelty of this paper consists in conducting the experiment aimed at development of the adaptive program with the help of machine learning and algebra-algorithmic methodology. The author uses algebraic techniques based on Systems of Algorithmic Algebra (SAA) being originated within the Ukrainian algebraic-cybernetic school [4, 5]. In previous works [3, 5] the facilities have been implemented in the instrumental software development system (called Integrated toolkit for Design and Synthesis of programs, IDS) that uses algebraic-algorithmic tools in three interdependent forms of representation of knowledge. The forms include analytical

CONCURRENCY, SPECIFICATION AND PROGRAMMING
M. Szczuka et al. (eds.): Proceedings of the international workshop CS&P 2011
September 28-30, Pułtusk, Poland, pp. 577-588

(algebraic formulae), natural language text and flowcharts. A particular feature and advantage of IDS is the interactive design of syntactically correct algorithms which is oriented toward elimination of syntax errors during program construction process. The primary goal of algebraic methodology is to produce software that is correct, safe and portable. However, the problem of increasing the adaptability of the programs being developed with algebra-algorithmic approach still remains. In this paper author carries out the experiment aimed at developing the example of adaptive sorting program. The development combines algebraic technique, machine learning and algorithm selection.

The outline of the paper is the following. In Section 1 the concept of algorithm selection problem and associated machine learning methods are considered. Section 2 is devoted to the results of conducted experiment.

## 1 Algorithm Selection Problem and Machine Learning

The algorithm selection problem aims at selecting the best algorithm for a given computational problem instance according to some characteristics of the instance. The formal abstract model of algorithm selection problem was proposed by J. Rice [13]. The mentioned model can be used to explore the question "With so many available algorithms, which one is likely to perform best on my problem and specific instance?". The main components of the model are the following:

- the problem space $P$ represents the set of instances of a problem class;
- the feature space $F$ contains measurable characteristics of the instances generated by a computational feature extraction process applied to $P$;
- the algorithm space $A$ is the set of all considered algorithms for tackling the problem;
- the performance space $Y$ represents the mapping of each algorithm to a set of performance metrics.

Then the algorithm selection problem can be formally stated as follows.

For a given problem instance $x \in P$, with features $f(x) \in F$, find the selection mapping $S(f(x))$ into algorithm space $A$, such that the selected algorithm $\alpha \in A$ maximizes the performance mapping $y(\alpha(x)) \in Y$.

The choice of features depends very much on the problem domain and the chosen algorithms. For example, when considering sorting algorithms for sequences of integers, the degree of presortedness of the input sequence is a problem feature. The algorithm execution time in seconds is the performance metric.

Using "if-then" rules the solution of algorithm selection problem can be formulated, for example, as follows:

*If input data have characteristics* $C_1$, $C_2$..., $C_n$,
*Then use algorithm* $A_1$ *instead of algorithm* $A_2$.

Algorithm selection can be either static or dynamic. Static algorithm selection system makes the selection and then commits to the selected algorithm, while

dynamic algorithm selection system may change its selection dynamically by monitoring the running of the algorithm.

In many real world situations, algorithm selection is done by hand by some experts who have a good theoretical understanding to the computational complexities of various algorithms and are very familiar with their runtime behaviors. One of the ways of automation of algorithm selection is to use machine learning [13].

Works [7, 9] are devoted to the automation of algorithm selection problem in the domain of sorting algorithms. In paper [9] authors expressed the task of sorting algorithm selection by considering the length of the sequence of integers to be sorted as the single feature, and using dynamic programming to estimate the mapping *S* via a Markov decision process. A larger study of sorting algorithms was conducted in [7] by H. Guo. He recognized that a sequence of integers to be sorted could be characterized by more than its length, but also by its degree of presortedness. Three measures of presortedness were used: the number of inversions (INV); number of runs of ascending subsequences (RUN); and the length of the longest ascending subsequence (LAS). He also showed that the most efficient is to use RUN metric.

The algorithm selection method is similar to metaheuristics [10, 15]. Metaheuristic is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. The candidate solution is a member of a set of possible solutions (e.g. a set of algorithms) to a given problem. Metaheuristics are designed to tackle complex optimization problems where other optimization methods have failed to be either effective or efficient [10]. Techniques which constitute metaheuristics algorithms range from simple local search procedures to complex learning processes.

This paper is devoted to the solution of algorithm selection problem for sorting, based on the approach proposed in previously mentioned work [7]. The main difference of this work consists in usage of algebra of algorithms for designing sorting programs, allowing to use high-level specifications which are not dependent on implementation in a specific programming language.

In work [7] algorithm selection problem is solved using a classification, which is one of the machine learning methods.

**Definition.** Let there is a set of objects; a *classification* is an algorithmic procedure that assigns any object from the set into one of a given number of classes [1, 2].

Let $X = \{x_1..., x_k\}$ be a set of attributes of an object, $Y = \{1, \ldots, m\}$ be a set of labels of classes. As a result of classification the *target function f* is received, which is a mapping from *X* to *Y*, $f: X \to Y$.

The target function is also known informally as a *classification model*.

For example, in case of algorithm selection problem for sorting the objects are processed arrays [7]. The attributes of objects are:

- $x_1$ – array size;
- $x_2$ – degree of presortedness.

The labels of classes are the names of various sorting algorithms (e.g. insertion sort, quick sort). The target function $f(x_1, x_2)$ defines for each array the name of the best algorithm to sort it (with regard to least execution time). In more details the example of the description of data for the given problem is considered in Section 2.

A classification technique (or classifier) is a systematic approach to building classification models from an input data set. The main steps of the classification usually are the following. The classifier is fed training data in which each object is already labeled with the correct class label. This data is used to train the learning algorithm, which creates classification models. The classification models then are used to classify similar data (test dataset).

Examples of classification techniques include decision tree classifiers, rule-based classifiers, neural networks, support vector machines, and naïve Bayes classification [2]. In work [7] it is shown, that the most efficient in case of algorithm selection problem for sorting is to use decision trees.

Decision tree learning is one of the most popular inductive learning methods [2]. It has been applied to a broad range of tasks from medical diagnosis to credit risk assessment. In decision tree learning, the learned function is represented by a decision tree. Decision trees are essentially sets of "if-then" rules. They classify training examples by sorting them down the tree from the root to some leaf node, which provides the classification of the example. Each node in the tree represents a test of some attribute of the training example, and each branch corresponds to one of the possible values for its source node (attribute). There are various algorithms for constructing decision trees: ID3, C4.5, NewId, ITrule, CN2, etc. [2].

Fig. 1 shows an example of the simplified decision tree for selection of one of the sorting algorithms depending on the size of the input array. The tree is constructed on the basis of the experimental data, obtained in work [9].

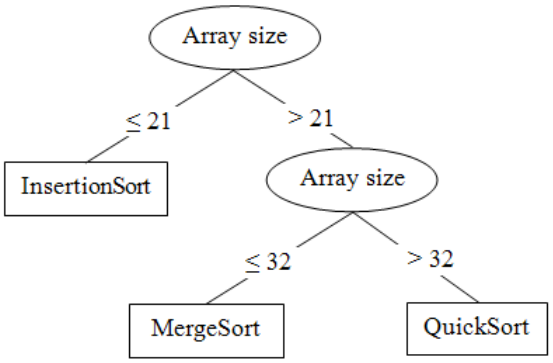


**Fig. 1.** Example of decision tree

In this paper, as well as in work [7], learning experiments are conducted in Weka, an open-source machine learning software in Java. Weka (Waikato Environment for Knowledge Analysis) [17] is a collection of machine learning algorithms for solving data mining problems. It supports data preprocessing, clustering, classification, regression, visualization, and feature selection. Weka is also well-suited for developing new machine learning schemes. All of Weka's techniques are predicated on the assumption that the data is available as a single table, where each data point is

described by a fixed number of attributes. The example of such table is considered in Subsection 2.2.

## 2 Development of Adaptive Sorting Algorithm

The goal of the experiments conducted in this work is to construct the adaptive sorting algorithm on the basis of several well-known algorithms. Namely, five algorithms were used: insertion sort, shell sort, heap sort, merge sort and quick sort [8]. By the adaptive sorting algorithm we mean an algorithm that takes advantage of size and existing order (presortedness) of its input. The experiment includes designing the initial algorithms with usage of algorithm algebras (see Subsection 2.1), and then using algorithm selection and machine learning methods to construct adaptive algorithm. The sorting algorithms were developed using the IDS toolkit [3, 5]. IDS supports automated designing of algebraic specifications of algorithms, called SAA schemes, and synthesis of code in programming languages (C++, Java).

The experiment consisted of the following stages:

1) formalized designing of initial set of sorting algorithms in systems of algorithmic algebra (see Subsection 2.1);

2) preparation of training data. At first, a set of input arrays with different characteristic values was generated. Then all sorting algorithms were executed on these arrays and the algorithm running time was collected. The algorithm that consumes the least time in sorting the arrays is labeled as the *best*. Based on the collected data, the table with the training data is composed, which includes the information about the size of each array, its presortedness degree and the best algorithm;

3) execution of machine learning algorithm (decision tree learning), on the training data with the help of Weka system;

4) transformation of the obtained decision tree to the SAA scheme of adaptive algorithm;

5) generation of programming code in C++ language based on SAA scheme;

6) comparison of execution time of the adaptive algorithm and initial algorithms on a test set of arrays.

Stages 2 to 6 are considered in more detail in Subsection 2.2.

### 2.1 Formalized Designing of Programs in Systems of Algorithmic Algebras

Sorting algorithms in this paper are designed using systems of algorithmic algebras [4, 5]. SAA is the two-based algebra $<\{U, B\}; \Omega>$, where $U$ is a set of logical conditions (predicates) and $B$ is a set of operators, defined on an informational set; $\Omega = \Omega_1 \cup \Omega_2$ is the signature of operations consisting of systems $\Omega_1$ and $\Omega_2$ of logical operations and operators respectively (these will be considered below).

Operator representations of algorithms in SAA are called regular schemes. The algorithmic language SAA/1 [5] is based on mentioned algebra and is used to describe algorithms in a natural language form. The algorithms, represented in SAA/1, are

called *SAA schemes*. The advantage of using SAA schemes is the ability to describe algorithms in a form suitable for a human facilitating achievement of demanded quality of programs.

Operators and predicates can be basic or compound. The basic operator (predicate) is the operator (predicate), which is considered in SAA schemes as primary atomic abstraction. The compound predicates are constructed from basic ones by logical SAA operations:

- disjunction: `'condition1' OR 'condition2'`;
- conjunction: `'condition1' AND 'condition2'`;
- negation: `NOT 'condition'`.

Compound operators are built from elementary ones by means of operations of serial and parallel execution operators:

- `"operator1" THEN "operator2"` is the serial execution of operators;
- `IF 'condition' THEN "operator1" ELSE "operator2" END IF` is the conditional execution of operators;
- `FOR 'condition' LOOP "operator1" END OF LOOP` is the for-loop;
- `WHILE NOT 'condition' LOOP "operator1" END OF LOOP` is the while-loop.

*Example 1*. The serial SAA scheme of insertion sort is given below. The algorithm sorts the input array *A* of size *n*. The identifiers of basic operators in the SAA scheme are written with double quotes and basic predicates are written with single ones.

```
SCHEME insertionSort(A, n)
==== FOR ' (i) from (1) to (n)'
     LOOP
        "(temp: = a[i])"
        THEN
        "(j: = i - 1)"
        WHILE NOT 'temp < a[j]'
                  AND
                  'j >= 0'
        LOOP
           "(a[j + 1] := a[j])"
           THEN
           "(j: = j - 1)"
        END OF LOOP
        "(a[j + 1] := temp)"
     END OF LOOP
END OF SCHEME insertionSort (A, n)
```

IDS system (see [3, 5]), developed by the author, is intended for interactive constructing of algorithm schemes in SAA and generating programs in programming languages (Java, C++). The main component of the system is the Constructor, which is intended to unfold designing algorithm schemes. The schemes are designed by superposition of SAA language constructs, which a user chooses from the list and

which are considered as reusable components for construction of algorithms. The design process is represented by an algorithm tree.

### 2.2 Experiment Results

For testing the performance of five sorting algorithms on various input data, the training set of integer arrays was generated. The set consisted of 800 arrays of size from 10 to 100 elements and contain the sequences of the following types:

- 500 arrays with randomly disordered elements. The random generation algorithm used is Algorithm 235 (Random Permutation) [6];
- 150 nearly-sorted arrays;
- 150 already sorted in reverse order arrays with $N$ permutations (with $N$ from 0.1 to 10% of the array size) [12].

For each array, the presortedness measure was computed and then all five sorting algorithms were executed on that array. The running time of each algorithm and the best algorithm, i.e., the one that takes the least time to sort the array, are recorded to a file.

The presortedness degree of array $A$ of size $n$ is computed according to the formula [7]

$$runs'(A) = \frac{runs(A)}{n},$$

where $runs(A)$ is the number of ascending substrings, or the “runs up”, of the array $A$. $runs'(A)$ takes values in a range (0 … 1].

For example, for the array

$$A = <|10 \mid 4\ 5\ 7 \mid 1\ 3 \mid 2\ 6\ 9 \mid 8\ |>,$$

$$runs(A) = 5;$$
$$runs' = 0.5.$$

For an already sorted sequence, $runs'(A) = 1 / n$, and for a sequence in reverse order, $runs'(A) = 1$.

The experimental data, collected during the execution of sorting programs, were used to generate the table with a training dataset (see Table 1). The dataset consists of three attributes: array size, presortedness degree ($runs'$) and the best algorithm, where best algorithm is the target attribute to be predicted based on other attributes of the array in question. As a result of the experiment only two of five considered algorithms (insertion sort and quick sort), appeared to be the best in various cases.

Distributions of size and presortedness degree of the training dataset, taking different values, are illustrated in Fig. 2 and Fig. 3 correspondingly.

**Table 1.** The fragment of the training dataset for selection of sorting algorithm

| Array number | Array size | Presortedness degree (*runs'*) | Best algorithm |
|---|---|---|---|
| 1 | 10.0 | 0.1 | insertion |
| 2 | 10.0 | 0.3 | insertion |
| 3 | 10.0 | 0.4 | insertion |
| 4 | 10.0 | 0.5 | insertion |
| 5 | 10.0 | 0.6 | insertion |
| 6 | 10.0 | 0.7 | insertion |
| 7 | 10.0 | 0.9 | quick |
| 8 | 10.0 | 1.0 | quick |

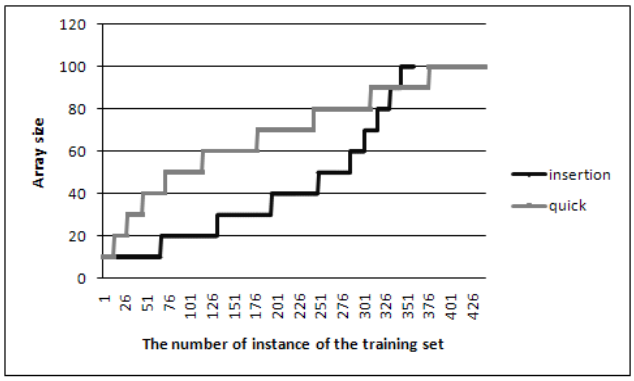


**Fig. 2.** The distribution of array size for training data with values “insertion” and “quick”

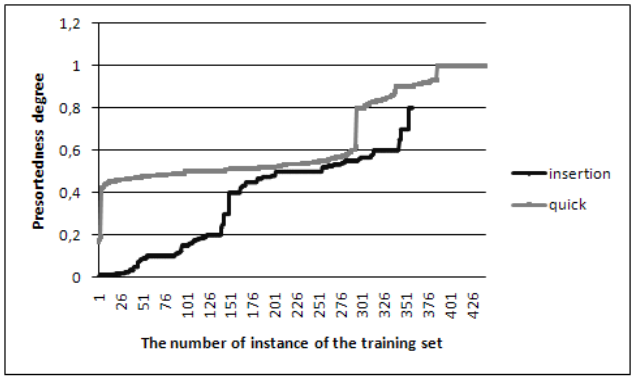


**Fig. 3.** The distribution of presortedness degree for training data with values “insertion” and “quick”

In Weka system, the decision tree algorithm J4.8 was applied to the obtained training data. The learned decision tree is shown in Fig. 4. The classification accuracy (percentage of correctly classified instances) of this tree is 93.625 %. The figure shows, in particular, that for small values of presortedness degree ($runs' \leq 0.4$), the best is the insertion sorting algorithm. In other cases the algorithm selection depends on the array size and *runs'*.

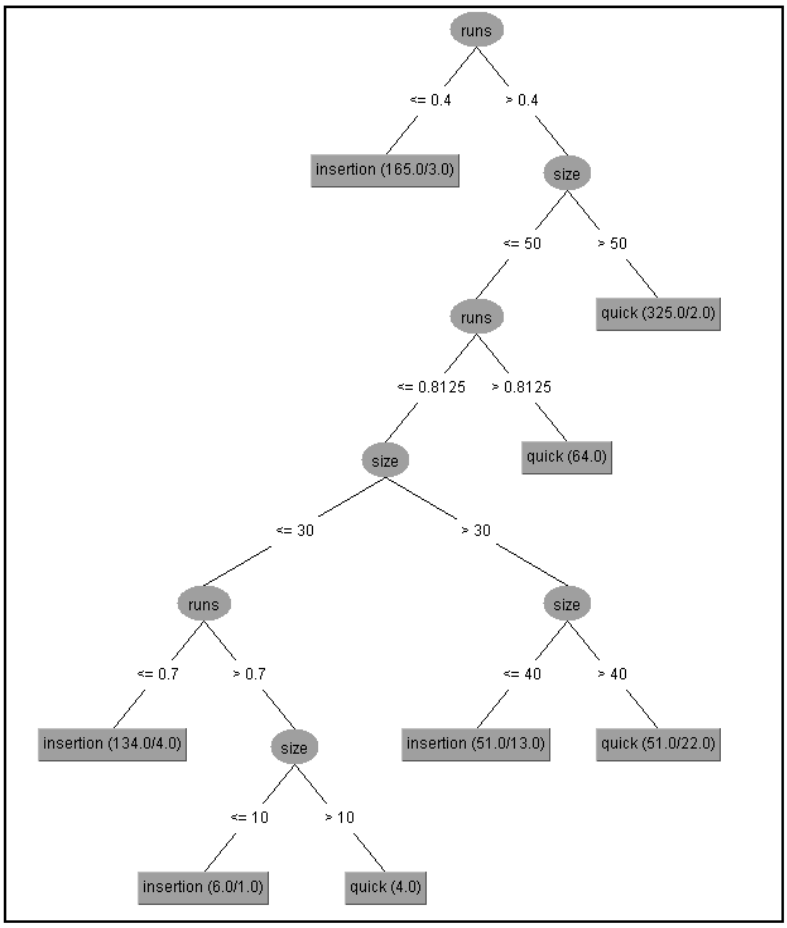


**Fig. 4.** The decision tree for selection of sorting algorithm

In the IDS toolkit, the decision tree was transformed to the SAA scheme of the adaptive algorithm `adaptiveSort(A, n)`, that is given below. The algorithm first computes the presortedness degree for the input array. Then, depending on the array size and *runs'* the corresponding algorithm (insertion sort or quick sort) is called. Further, IDS was used for generation of programming code in C++ language for this algorithm.

```
SCHEME adaptiveSort(A, n)
====
"(runs := "Compute the presortedness degree for array (A)
  of size (n)")"
IF ‘runs <= 0.4’ THEN "insertionSort(A, n)"
ELSE
   IF ‘runs > 0.4’ THEN
   IF ‘size <= 50’ THEN
      IF ‘runs <= 0.8125’ THEN
         IF ‘size <= 30’ THEN
            IF ‘runs <= 0.7’ THEN "insertionSort(A, n)"
            ELSE IF ‘runs > 0.7’ THEN
                     IF ‘size <= 10’ THEN "insertionSort(A, n)"
                     ELSE IF ‘size > 10’ THEN "quickSort(A, 0, n-1)"
                           END IF
                     END IF
                  END IF
            END IF
         ELSE IF ‘size > 30’ THEN
                  IF ‘size <= 40’ THEN "insertionSort(A, n)"
                  ELSE IF ‘size > 40’ THEN "quickSort(A, 0, n-1)"
                        END IF
                  END IF
               END IF
         END IF
      ELSE IF ‘runs > 0.8125’ THEN "quickSort(A, 0, n-1)"
            END IF
      END IF
   ELSE IF ‘size > 50’ THEN "quickSort(A, 0, n-1)"
         END IF
   END IF
   END IF
END IF
END OF SCHEME adaptiveSort(A, n)
```

For verifying the efficiency of the obtained adaptive algorithm, a test set of input arrays was prepared. It consisted of 140 integer arrays of size 100 and included the same type of sequences as the training set of arrays (randomly generated, nearly-sorted and reverse order arrays). An experiment was carried out on Intel Core 2 Quad CPU, 2.51 GHz, Windows XP machine. Fig. 5 shows the total execution time in microseconds of each sorting algorithm (first five bars) and the adaptive algorithm (the last bar), that were applied on the test set. The adaptive algorithm outperforms all sorting algorithms, that is the evidence of efficiency of the approach, proposed in this work.

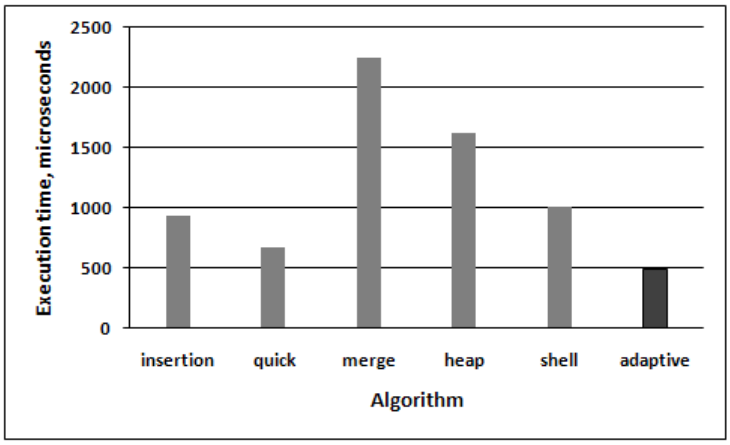


**Fig. 5.** The total time spent by each sorting algorithm on the test set of arrays

## 6 Conclusion

The experiment aimed at development of adaptive sorting program on the basis of usage of algorithm selection method, machine learning system and algebra-algorithmic approach is conducted. Machine learning facilities allow to automatize constructing of adaptive algorithm on the basis of analysis of experimental data, related to execution of initial algorithms. Designing of algorithms is based on usage of systems of algorithmic algebras. The advantages of using SAA schemes are their simplicity, independence from programming language and possibility of translation to arbitrary programming language. The experiment showed better performance of the developed adaptive algorithm as compared with initial sorting algorithms, which is the evidence of the efficiency of the proposed approach.

The prospects of further investigations in this direction are integration of IDS and Weka systems, and also applying of the proposed approach for other subject domains.

## References

1. Classification in machine learning. – http://en.wikipedia.org/wiki/Classification_in_machine_learning
2. Classification: Basic Concepts, Decision Trees, and Model Evaluation. – www-users.cs.umn.edu/~kumar/dmbook/ch4.pdf
3. Doroshenko, A., Kotyuk, M., Nikolayev, S., Tseytlin, G., Yatsenko, O.: Developing Parallel Programs with Algebra of Algorithms and Heuristic Facilities. Proc. Int. Workshop

"Concurrency: Specification and Programming" (CS&P 2009), Kraków-Przegorzały, Poland, 28–30 September 2009 (2009) 142–153
4. Doroshenko, A., Tseytlin, G., Yatsenko, O., Zachariya, L.: A Theory of Clones and Formalized Design of Programs. Proc. Int. Conf. "Concurrency, Specification and Programming" (CS&P'2006), 27–29 September 2006, Wandlitz, Germany (2006) 328–339
5. Doroshenko, A., Tseytlin, G., Yatsenko, O., Zachariya, L.: Intensional Aspects of Algebra of Algorithmics. Proceedings of International Workshop "Concurrency, Specification and Programming" (CS&P'2007), 27–29 September 2007, Lagow (Poland) (2007)
6. Durstenfeld, R.: Algorithm 235: Random permutation. Communications of the Association for Computing Machinery, 7:420 (1964)
7. Guo, H.: Algorithm selection for sorting and probabilistic inference: A machine learning-based approach. Ph.D. dissertation, Kansas State University (2003)
8. Knuth, D.E.: The art of computer programming: Sorting and Searching, volume 3. Addison-Wesley (1981)
9. Lagoudakis, M., Littman, M., Parr, R.: Selecting the right algorithm. In Proceedings of the AAAI Fall Symposium Series: Using Uncertainty within Computation (2001)
10. Olafsson, S.: Metaheuristics. – http://www.public.iastate.edu/~olafsson/metaheuristics.pdf
11. Samulowitz, H., Memisevic, R.: Learning to solve QBF. In Proceedings of the 22nd AAAI Conference on Artificial Intelligence (2007) 255–260
12. Slightly Skeptical View on Sorting Algorithms. – http://www.softpanorama.org/Algorithms/sorting.shtml
13. Smith-Miles, K.A.: Cross-Disciplinary perspectives on meta-learning for algorithm selection. ACM Comput. Surv., 41, 1, Article 6 (December 2008) (2008) 25 p.
14. Smith-Miles, K.A.: Towards insightful algorithm selection for optimization using meta-learning concepts. In Proceedings of the IEEE Joint Conference on Neural Networks (2008) 4118–4124
15. Tseng, L.Y.: Metaheuristic Methods and Their Applications. – http://163.17.20.188/IAE/manager/intraspeech/file/file_7.pdf
16. Wang, X., Smith, K.A., Hyndman, R.: Characteristic-Based clustering for time series data. Data Mining Knowl. Discov. 13 (2006) 335–364
17. Weka Project home page. – http://www.cs. waikato.ac.nz/ml/weka/
18. Witten, I.H., Frank, E., Hall, M.A.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann (2011) 629 p.