A new model for classifying DNA code inspired by neural networks and FSA

Byeong Kang, Andrei Kelarev, Arthur Sale, Ray Williams

School of Computing
University of Tasmania
Private Bag 100, Hobart
Tasmania 7001, Australia
{*BHKang,Andrei.Kelarev,Arthur.Sale,R.Williams*}*@utas.edu.au*
`www.comp.utas.edu.au/users/{bhkang/,kelarev/,ahjs/,rwilliams/}`

Abstract. This paper introduces a new model of classifiers $\mathrm{CL}(V, E, \ell, r)$ designed for classifying DNA sequences and combining the flexibility of neural networks and the generality of finite state automata. Our careful and thorough verification demonstrates that the classifiers $\mathrm{CL}(V, E, \ell, r)$ are general enough and will be capable of solving all classification tasks for any given DNA dataset. We develop a minimisation algorithm for these classifiers and include several open questions which could benefit from contributions of various researchers throughout the world.

1 Introduction

Classification of data is important in data mining, see [39]. The results of this paper make the very first essential and rather non-trivial step of work on IRGS grant allocated by the University of Tasmania for the development and investigation of new Artificial Intelligence methods for classification of DNA data collected by the School of Plant Science and CRC for Sustainable Production Forestry. This is why we are mainly interested in DNA sequences, and we record all new definitions in this case. In fact, the results and concepts of this note are applicable to larger classes of problems and can be used to classify texts and documents, see for example [3], [4], [11], [21], [22], [23], [24], [27], [28], [29], [30], [32], as well as sequences in datasets of various other kinds too.

The applications of neural networks to solving numerous practical tasks have been very well known. Many useful results have been obtained with neural networks in various applied branches. For the purposes of classifying DNA sequences it is impossible to use neural networks directly processing the sequences of nucleotides. As a guide we have to look at another very well known concept of a finite state automaton (FSA) used for analysing sequences. We refer to [6], [9], [13], [14], [15], [16], [17], [19], [20], [31], [33], [34], [38] for background and some relevant recent results on the subject. The first aim of the present paper is to generalize the architecture of neural networks in order to encompass all FSA in a new concept.

Let us begin by introducing a new model of classifiers $\mathrm{CL}(V, E, \ell, r)$ as a simultaneous generalisation of neural networks and finite state automata. This model combines the flexibility of neural networks and the generality of finite state automata. It is likely that this new notion will attract the attention of researchers. We develop a minimisation algorithm for the classifiers $\mathrm{CL}(V, E, \ell, r)$. This paper includes several challenging open questions, which could benefit from contributions of many investigators throughout the world.

Before the start of experimental investigation, first of all it is important to demonstrate that the model is suitable for handling sufficiently general classes of problems and can avoid pitfalls. The main result of this paper provides the readers with a thorough verification of the fact that the classifiers $\mathrm{CL}(V, E, \ell, r)$ are capable of handling all classification problems for DNA sequences.

Our formal model is also related to the more general concept of a labeled graph. Labeled graphs have valuable applications in various areas and have been investigated by many researchers too. We refer to [7] for a dynamic survey on graph labeling available online from the Electronic Journal of Combinatorics (see also, for instance, [36] and [37]).

The notion of classifiers $\mathrm{CL}(V, E, \ell, r)$ has been carefully chosen from the very beginning to combine the generality of finite state automata and the flexibility of neural networks. Our main theorem shows that the classifiers $\mathrm{CL}(V, E, \ell, r)$ can handle all classification problems for DNA datasets given sufficient computing time. A separate section develops a minimisation algorithm for these classifiers.

Background information and preliminaries are included in Section 2. Our new model is defined in Section 3. Section 4 contains the main theorem. Several major differences between classifiers $\mathrm{CL}(V, E, \ell, r)$, neural networks, and finite state automata are pointed out in Section 5. A minimization algorithm for classifiers $\mathrm{CL}(V, E, \ell, r)$ is presented in Section 6. Open questions are collected in Section 7.

2 Preliminaries

We use standard concepts concerning graphs and algorithms, following [2] and [35]. Throughout the word 'graph' will mean a directed graph, which is allowed to have multiple edges and loops. We refer to [8] for preliminaries on algorithms for computational analysis of DNA sequences.

Let us refer to the monographs Baldi and Brunak [1], Durbin, Eddy, Krogh and Mitchison [5], Jones and Pevzner [10] and Mount [26] for preliminaries on bioinformatics. Here we briefly recall that every DNA molecule is a double helix consisting of two strands. Each strand is a sequence of 4 nucleotides or bases: A (adenine), C (cytosine), G (guanine), and T (thymine). According to the Watson-Crick complementarity each nucleotide in one strand is crosslinked to a complementary nucleotide in another strand, and together they form a base pair. For example, the human genome contains about 3 billion base pairs and about 35,000 genes. In each DNA molecule, A and T always complement each other: A in one strand is linked to T in the second spiral. Similarly, C and G complement each other: C in one spiral is always linked to G in another strand.

If we know one sequence, it's easy to determine its complement. Therefore the sequence of base pairs in every DNA molecule can be represented with just one string over the alphabet of four letters A,C,G,T. In this paper we consider the problem of classifying strings of letters over the alphabet

$$X = \{A, C, G, T\}.$$

Accordingly, the set of all DNA sequences is precisely the set X^* of all strings over X.

3 Main Notion

A *classifier* $\mathrm{CL}(V, E, \ell, r)$ is a quadruple

$$\mathrm{CL}(V, E, \ell, r) = (V, E, \ell, r), \tag{1}$$

where $V = \{v_1, \ldots, v_n\}$ is the set of vertices and E is the set of edges of a graph $G = (V, E)$ with multiple edges allowed and with each edge e labeled by a letter $\ell(e)$ of the alphabet X and a real number $r(e)$. In other words, there are two functions

$$\ell : E \to X \text{ and } r : E \to \mathbb{R}. \tag{2}$$

The *state* (or *current state*) of the classifier $\mathrm{CL}(V, E, \ell, r)$ is a labeling of all vertices by real numbers, i.e., a function

$$s : V \to \mathbb{R}. \tag{3}$$

Notice that our model has some similarities with the concept of a finite state automaton and that of a neural network, but is different from them.

The classifiers $\mathrm{CL}(V, E, \ell, r)$ potentially can be used for both classification and clustering. A classification of any given set of DNA sequences is a partition of these sequences into several classes. Classifiers obtain classifications via various algorithms for supervised learning. In this way the classification is known for the given set of data. The problem is to construct a classifier that will produce this classification, so that it can then be used to determine class membership of new sequences. Initial partition is usually communicated by a supervisor to a machine learning process constructing the classifier. A different problem is that of clustering data. It deals with dividing a set of given sequences into classes not known initially, but determined according to certain measures of similarities between sequences. This is usually accomplished via a process of unsupervised learning, see [39].

Now suppose that we want to use the classifier $\mathrm{CL}(V, E, \ell, r)$ to analyse a DNA sequence

$$x_1, x_2, \ldots, x_N, \tag{4}$$

where $x_1, \ldots, x_N \in X$. The initial state $s_0 : V \to \mathbb{R}$ can be chosen arbitrarily depending on practical implementation. Then we use the labeled graph to recursively process all letters of the sequence (4) and modify the state of the graph.

Suppose that after we have considered the first $i \geq 0$ letters of (4) the state of the graph is

$$s_i : V \to \mathrm{I\!R}.$$

Then we can determine the next state s_{i+1} with recursion

$$s_{i+1}(v) = \sum_{w \in V, (w,v) \in E} r((w,v)) s_i(w). \tag{5}$$

After the whole sequence (4) has been processed, for every vertex $v \in V$, we know the final value $s_N(v) \in \mathrm{I\!R}$.

Let us now define the standard partitions which we are going to use in classification of DNA sequences. The following standard partitions will be associated with the classifier $\mathrm{CL}(V, E, \ell, r)$. For every $1 \leq k \leq N$, we define the classification $\mathcal{K}_k$ as the one which divides all given DNA sequences into classes $C_1, \dots, C_k$, by including the sequence (4) into the class $C_i = C_i^{(k)}$, where i is chosen so that $1 \leq i \leq k$, and

$$s_N(v_i) = \max\{s_N(v_1), \dots, s_N(v_k)\}.$$

Obviously, for $k > 1$, every classification $\mathcal{K}_k$ can be obtained from $\mathcal{K}_{k-1}$ by selecting certain elements in all classes

$$C_1^{(k-1)}, C_2^{(k-1)}, \dots, C_{k-1}^{(k-1)}$$

of $\mathcal{K}_{k-1}$ and including them in the new class $C_k^{(k)}$. Thus, every previous classification can be regarded as a simplified version of the next one, and every next classification is a refinement of the preceding one.

4 Main Result and Verification

The main theorem of this paper establishes that the classifiers $\mathrm{CL}(V, E, \ell, r)$ are capable of solving all classification tasks for any given dataset of DNA sequences.

Theorem 1. *For each set S of DNA sequences and every given partition*

$$S = S_1 \dot\cup S_2 \dot\cup \cdots \dot\cup S_k \tag{6}$$

one can find a classifier $\mathrm{CL}(V, E, \ell, r)$

$$C = (V, E, \ell, r) \tag{7}$$

which produces classification

$$\mathcal{K} : X^* = C_1 \dot\cup C_2 \dot\cup \cdots \dot\cup C_k \tag{8}$$

such that the classes of partition (6) *are determined by the classes of classification* (8) *so that $S_i = S \cap C_i$ for all $i = 1, \dots, k$.*

Proof. First, let us define convenient notation which will enable us to refer to all sequences and their base pairs. Putting $N = |S|$, denote the sequences of the set S by $b^{(1)}, b^{(2)}, \ldots, b^{(N)}$. For each $i = 1, \ldots, N$, denote the bases of the sequence $b^{(i)}$ by the symbols $b_j^{(i)}$, where $j = 1, \ldots, m_i$ so that

$$b^{(i)} = b_1^{(i)}, b_2^{(i)}, \ldots, b_{m_i}^{(i)},$$

for all $i = 1, \ldots, N$. Suppose that the sequence $b^{(i)}$ belongs to the class $S_{\phi(i)}$ of partition (6), where ϕ is a function from $[1 : N]$ into $[1 : k]$.

Next, we introduce the following sets of vertices for the classifier $\mathrm{CL}(V, E, \ell, r)$ we are going to construct:

$$V_0 = \{v_1, v_2, \ldots, v_k\} \tag{9}$$

$$V_i = \{v_1^{(i)}, v_2^{(i)}, \ldots, v_{m_i-1}^{(i)}\} \tag{10}$$

for $i = 1, 2, \ldots, N$. In addition, choose a vertex v_0 which does not belong to any of the sets $V_0, V_1, \ldots, V_N$ and suppose that these sets are pairwise disjoint and all of their vertices are distinct. Put

$$V = V_0 \cup V_1 \cup \cdots \cup V_N \cup \{v_0\}. \tag{11}$$

To simplify further notation and have uniform definitions, we are going to denote one and the same vertex v_0 by several alternative symbols $v_0^{(1)}, v_0^{(2)}, \ldots, v_0^{(N)}$ too. Similarly, for $i = 1, 2, \ldots, N$, we introduce a new symbol $v_{m_i}^{(i)}$ to be used as an alternative notation for the vertex $v_{\phi(i)} \in V_0$. For $i = 1, 2, \ldots, N$, let us introduce sets of edges

$$E_i = \{(v_0, v_1^{(i)}) = (v_0^{(i)}, v_1^{(i)}), (v_1^{(i)}, v_2^{(i)}), \ldots, (v_{m_i-1}^{(i)}, v_{m_i}^{(i)}) = (v_{m_i}^{(i)}, v_{\phi(i)})\} \tag{12}$$

and put

$$E = E_1 \cup E_2 \cup \ldots \cup E_N. \tag{13}$$

It remains to define the initial state s_0 and the labels ℓ and r, see (2) and (3). For all $i = 1, 2, \ldots, N$ and $j = 1, 2, \ldots, m_i$, put

$$\ell((v_{j-1}^{(i)}, v_j^{(i)})) = b_j^{(i)}, \tag{14}$$

$$r((v_{j-1}^{(i)}, v_j^{(i)})) = 1. \tag{15}$$

The initial state s_0 is defined by putting, for $v \in V$,

$$s_0(v) = \begin{cases} 1 & \text{if } v = v_0, \\ 0 & \text{otherwise.} \end{cases} \tag{16}$$

This completes the definition of the classifier $\mathrm{CL}(V, E, \ell, r)$.

Suppose that the classifier is used to process the sequence $b^{(i)}$, where $1 \le i \le N$. We are going to show by induction that after considering the first j bases of the sequence the current state of the classifier will satisfy

$$s_j(v) = \begin{cases} 1 & \text{if } v = v_j^{(i)}, \\ 0 & \text{otherwise,} \end{cases} \tag{17}$$

for any $v \in V$.

The induction basis is provided by (16). Suppose that equalities (17) have been established for some $1 < j < m_i$. Then we can find the next state $s_{j+1}(v)$ using recursion (5).

First, consider the case where $v = v^{(i)}_{j+1}$. Since E contains only one edge of the form $(w, v^{(i)}_{j+1})$, and $s_j(v^{(i)}_j) = 1$ by the induction assumption, (15) and (5) yield us that

$$\begin{aligned}
s_{j+1}(v) &= \sum_{w \in V, (w,v) \in E} r((w,v)) s_j(w) && (18)\\
&= \sum_{w \in V, (w,v^{(i)}_{j+1}) \in E} r((w, v^{(i)}_{j+1})) s_j(w) && (19)\\
&= r((v^{(i)}_j, v^{(i)}_{j+1})) s_j(v^{(i)}_j) && (20)\\
&= s_j(v^{(i)}_j) && (21)\\
&= 1. && (22)
\end{aligned}$$

Thus, for $v = v^{(i)}_{j+1}$, the required version of (17) holds indeed.

Second, assume that $v \neq v^{(i)}_{j+1}$. Consider any $w \in V$. If $w = v^{(i)}_j$, then $(w, v) \notin E$ by the choice of v. If, however, $w \neq v^{(i)}_j$, then $s_j(w) = 0$ by the induction assumption. Thus all summands in recursion (5) vanish and we get

$$\begin{aligned}
s_{j+1}(v) &= \sum_{w \in V, (w,v) \in E} r((w,v)) s_j(w) && (23)\\
&= 0. && (24)
\end{aligned}$$

This means that the desired version of (17) holds if $v \neq v^{(i)}_{j+1}$, too. By the principle of mathematical induction, it follows that (17) is always satisfied.

After all bases $b^{(i)}_1, \ldots, b^{(i)}_{m_i}$ of the sequence $b^{(i)}$ have been processed, the final state of the the classifier $\mathrm{CL}(V, E, \ell, r)$ turns into

$$s_{m_i}(v) = \begin{cases} 1 & \text{if } v = v^{(i)}_{m_i} = v_{\phi(i)}, \\ 0 & \text{otherwise.} \end{cases} \qquad (25)$$

According to our definition $b^{(i)}$ belongs to the class $C_{\phi(i)}$ of the classification $\mathcal{K}_k$, as required. This means that our classifier indeed produces a classification that agrees with the given partition of data, and so the proof is complete.

5 Neural Networks and Finite State Automata

Let us begin by comparing the classifiers $\mathrm{CL}(V, E, \ell, r)$ with neural networks. Neural networks can be represented with similar labeled graphs. In this case the vertices are called neurons, and the labels of the edges are called weights.

Edge labels are modified while a neural network is being trained. After that during the operation of the network the labels remain unchanged. Each neuron of the network takes a weighted sum of its inputs and passes it through a threshold function, usually the sigmoid function. As indicated above, the classifiers $\mathrm{CL}(V,E,\ell,r)$ are different from neural networks and finite state automata.

The major difference is that neural networks and classifiers $\mathrm{CL}(V,E,\ell,r)$ are designed to solve substantially different types of problems. Neural networks cannot be directly applied to classification of DNA sequences without collections of some additional data, for example, from microarrays. The reason for this is that the operation of every neural network depends on a relatively small number of input parameters, represented as continuous real values. Small changes to the values of these parameters are not generally supposed to create changes to the classification outcome. Hence it is impossible to encode whole long DNA sequences in this way. In contrast, the classifiers $\mathrm{CL}(V,E,\ell,r)$ can process all base pairs of a given DNA sequence in succession.

Sophisticated continuous threshold functions used in neural networks lead to another serious difference (see [25], Section 11). Although the current state of a classifier $\mathrm{CL}(V,E,\ell,r)$ appears similar to the state of a neural network, the transition to the next state is accomplished in a completely different fashion.

Comparing the classifiers $\mathrm{CL}(V,E,\ell,r)$ to finite state automata, let us just note that each finite state automaton is used to divide its input into two classes only. Besides, the edges of finite state automata do not have real numbers as labels. These labels are inspired by analogy with neural networks. They make classifiers $\mathrm{CL}(V,E,\ell,r)$ more flexible than finite state automata. This is why it is natural to expect that future research will demonstrate the possibility of substantial reduction to the size of the classifiers $\mathrm{CL}(V,E,\ell,r)$ designed to handle certain classification tasks.

6 Main Algorithm

After a classifier $\mathrm{CL}(V,E,\ell,r)$ has been found, the next natural step is to make it smaller. This can be achieved by identifying equivalent vertices. We say that a classifier $\mathrm{CL}(V,E,\ell,r)$ is *minimal* if it can no longer be simplified by combining and identifying its vertices in some groups. As a guide to developing our minimization algorithm we are going to use the established standard terminology for analogous situations known in automata theory. Our new algorithm originates from the reduction algorithm for finite state automata described in several books (see, for example, [13], Section 3.7).

The minimization algorithm we are going to develop applies only to classifiers of the special type used in the proof of our main theorem. Namely, here we restrict our attention to the classifiers where each current state is a characteristic function of one of the vertices: it is equal to 1 at this vertex, and is equal to 0 at all other vertices. The special vertex will be called the *vertex* of the current state.

The algorithm proceeds by identifying equivalent vertices, so that one can combine them without affecting the action of the classifier $\mathrm{CL}(V, E, \ell, r)$ on input strings.

The concepts of equivalence and congruence will be used in order to simplify the classifiers $\mathrm{CL}(V, E, \ell, r)$. They formalise and provide exact meaning to the idea of dividing all vertices of a the classifier $\mathrm{CL}(V, E, \ell, r)$ into groups in such a way that the operation of the classifier remains unchanged if all vertices in each group are regarded as one new vertex.

Consider a classifier $C = (V, E, \ell, r)$. In order to define the concept of a congruence on C, we begin with a few auxiliary notions. First of all, let us recall the definition of an equivalence relation. It is required, because every partition of the set of vertices V into classes can be achieved using an equivalence relation. Every subset of the set

$$V \times V = \{(u, v) \mid u, v \in V\} \tag{26}$$

is called a *relation* on the set V of all vertices. A relation ϱ is said to be *symmetric* if $(u, v) \in \varrho$ implies $(v, u) \in \varrho$, for all $u, v \in V$. It is *transitive* if $(u, v), (v, w) \in \varrho$ implies $(u, w) \in \varrho$, for all $u, v, w \in V$. A relation is said to be *reflexive* if it contains the set

$$\{(v, v) \mid v \in V\}. \tag{27}$$

An *equivalence* relation is a relation which is reflexive, symmetric and transitive.

If ϱ is a relation on V and $v \in V$, then we put

$$v^{\varrho} = \{w \mid (v, w) \in \varrho\}. \tag{28}$$

The set v^{ϱ} is called the equivalence class of ϱ containing v. It is known and easy to verify that ϱ is an equivalence relation if and only if the sets v^{ϱ}, $v \in V$, form a partition of V into several equivalence classes.

Let ϱ be an equivalence relation on C. Next, we show how ϱ simplifies C by combining all vertices which belong to the same classes of ϱ. The resulting classifier will be called a quotient classifier. Namely, the *quotient classifier* C/ϱ is the quadruple

$$C/\varrho = (V/\varrho, E/\varrho, \ell/\varrho, r/\varrho), \tag{29}$$

where the sets V/ϱ, E/ϱ and functions ℓ/ϱ, r/ϱ are defined as follows. The set V/ϱ is the set of all equivalence classes of ϱ on V. The set E/ϱ will contains an edge $(u^{\varrho}, v^{\varrho})$ with

$$(\ell/\varrho)((u^{\varrho}, v^{\varrho})) = x \in X \tag{30}$$

if and only if there exist $u' \in u^{\varrho}$ and $v' \in v^{\varrho}$ such that $(u', v') \in E$ and $\ell((u', v')) = x$. In this case we set

$$(r/\varrho)((u^{\varrho}, v^{\varrho})) = \sum_{u' \in u^{\varrho}, v' \in v^{\varrho}, (u', v') \in E, \ell((u', v')) = x} r((u', v')). \tag{31}$$

To simplify notation, we will use the same symbols ℓ and r for the functions ℓ/ϱ and r/ϱ, too.

We say that two vertices of a the classifier $\mathrm{CL}(V, E, \ell, r)$ are **-equivalent* if the result of classification of each word by the classifier $\mathrm{CL}(V, E, \ell, r)$ starting in the state of one of vertices coincides with its classification result when it starts from the state of the second vertex.

In order to determine whether two vertices are *-equivalent, the algorithm uses an iterative process based on k-equivalence. Two states are said to be k-*equi*valent if every word of length $\leq k$ produces identical classification outcomes in the case where the classifier $\mathrm{CL}(V, E, \ell, r)$ starts in the state of the first vertex, exactly as when it starts in the second vertex. It is straightforward to verify that *-equivalence is a congruence.

In order to start the process, let us say that two vertices s and t of the the classifier $\mathrm{CL}(V, E, \ell, r)$ $C = (V, E, \ell, r)$ are 0-*equivalent* to each other if and only if they coincide. Next, suppose that for some $k \geq 0$ the k-equivalence has already been defined. Taking any two vertices s and t in V, we say that s is $(k + 1)$-equivalent to t if and only if s and t are k-equivalent and, for each input letter $x \in X$, if the classifier starts in the state of the vertex s and processes the letter x, then it arrives at exactly the same state that is achieved if it starts in the state of the vertex t and processes the letter x from that state, so that there is no difference between starting from s or from t.

The method of computing the k-equivalence classes from $(k - 1)$-equivalence classes is a dynamic programming algorithm. It finds the k-equivalence classes by subdividing the $(k - 1)$-equivalence classes according to the change of state of the classifier $\mathrm{CL}(V, E, \ell, r)$ when it reads each of the letters in X.

Since the set of all vertices is finite, they cannot be combined indefinitely, and at some stage the algorithm terminates. For some integer $k \geq 0$, the set of k-equivalence classes will coincide with the set of $(k + 1)$-equivalence classes. At this stage we see that both k-equivalence and $(k + 1)$-equivalence are in fact the $*$-equivalence.

These explanations show that the following steps find a minimal classifier $\mathrm{CL}(V, E, \ell, r)$ equivalent to the original one:

1. Find the set of 0-equivalence classes of V.

2. For $k = 0, 1, 2$, and so on, if k-equivalence classes have been found, then divide them as described above to find the $(k + 1)$-equivalence classes of V. Stop when the set of $(k + 1)$-equivalence classes is equal to the set of k-equivalence classes, for some integer k. This step gives the set of $*$-equivalence classes, as explained above.

3. Construct the minimal classifier $\mathrm{CL}(V, E, \ell, r)$ by identifying all vertices of the classes of $*$-equivalence.

7 Open Questions

Problem 1. Develop a minimization algorithm for classifiers $\mathrm{CL}(V, E, \ell, r)$ with arbitrary current state functions.

Problem 2. Evaluate the running time and develop more efficient minimization algorithms for these classifiers.

Problem 3. Develop more robust algorithms by introducing a preprocessing step which will augment the dataset with other sequences and achieve similar classifications for sequences which are similar.

Two other related models used in the analysis of DNA sequences are Markov Models and probabilistic finite state automata, see Baldi and Brunak [1], Durbin, Eddy, Krogh and Mitchison [5], Jones and Pevzner [10] and Mount [26]. They have been used to identify and classify segments of one DNA sequence and are different from our model. It may make sense to explore the possibility of using these notions to classify sets of whole large DNA sequences too. This leads to the following questions suggested by the referees of this paper.

Problem 4. Investigate the running times and compare the classifications produced by our new model with those which can be obtained using Markov Models.

Problem 5. Investigate the running times and compare the classifications produced by our new model with those which can be obtained using probabilistic finite state automata.

8 Acknowledgements

This research has been supported by the IRGS grant K14313 of the University of Tasmania and Discovery grant DP0449469 from the Australian Research Council.

The authors are grateful to the referees for suggesting interesting open questions recorded in Probems 4 and 5.

References

1. Baldi, P. and Brunak, S.: "Bioinformatics : The Machine Learning Approach". Cambridge, Mass, MIT Press, (2001).
2. Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C.: "Introduction to Algorithms", The MIT Press, Cambridge, 2001.
3. Dazeley, R.P., Kang, B.H.: Weighted MCRDR: deriving information about relationships between classifications in MCRDR, AI 2003: Advances in Artificial Intelligence, Perth, Australia, 2003, 245–255.
4. Dazeley, R.P., Kang, B.H.: An online classification and prediction hybrid system for knowledge discovery in databases, Proc. AISAT 2004, The 2nd Internat. Conf. Artificial Intelligence in Science and Technology, Hobart, Tasmania, 2004, 114–119.
5. Durbin, R., Eddy, S.R., Krogh, A. and Mitchison, G.: "Biological Sequence Analysis". Cambridge University Press (1999).
6. Eilenberg, S.: "Automata, Languages, and Machines". Vol. A,B, Academic Press, New York, 1974.
7. Gallian, J.A.: Graph labeling, Electronic J. Combinatorics, Dynamic Survey DS6, January 20, 2005, 148pp, www.combinatorics.org
8. Gusfield, D.: "Algorithms on Strings, Trees, and Sequences", Computer Science and Computational Biology, Cambridge University Press, Cambridge, 1997.

9. Holub, J., Iliopoulos, C.S., Melichar, B. Mouchard, L.: Distributed string matching using finite automata, “Combinatorial Algorithms”. AWOCA 99, Perth, 114–127.
10. Jones, N.C. and Pevzner, P.A.: An Introduction to Bioinformatics Algorithms. Cambridge, Mass, MIT Press, (2004). http://www.bioalgorithms.info/
11. Kang, B.H.: “Pacific Knowledge Acquisition Workshop”. Auckland, New Zealand, 2004.
12. Kelarev, A.V.: “Ring Constructions and Applications”. World Scientific, 2002.
13. Kelarev, A.V.: “Graph Algebras and Automata”. Marcel Dekker, 2003.
14. Kelarev, A.V., Miller, M. and Sokratova, O.V.: Directed graphs and closure properties for languages. “Proc.12 Australasian Workshop on Combinatorial Algorithms” (Ed. E.T. Baskoro), Putri Gunung Hotel, Lembang, Bandung, Indonesia, July 14–17, 2001, 118–125.
15. Kelarev, A.V., Miller, M. and Sokratova, O.V.: Languages recognized by two-sided automata of graphs. Proc. Estonian Akademy of Science **54** (2005) (1), 46–54.
16. Kelarev, A.V. and Sokratova, O.V.: Languages recognized by a class of finite automata. Acta Cybernetica **15** (2001), 45–52.
17. Kelarev, A.V. and Sokratova, O.V.: Directed graphs and syntactic algebras of tree languages. J. Automata, Languages & Combinatorics **6** (2001)(3), 305–311.
18. Kelarev, A.V. and Sokratova, O.V.: Two algorithms for languages recognized by graph algebras. Internat. J. Computer Math. **79** (2002)(12) 1317–1327.
19. Kelarev, A.V. and Sokratova, O.V.: On congruences of automata defined by directed graphs. Theoret. Computer Science **301** (2003), 31–43.
20. Kelarev, A.V. and Trotter, P.G.: A combinatorial property of automata, languages and their syntactic monoids. Proceedings of the Internat. Conf. Words, Languages and Combinatorics III, Kyoto, Japan, 2003, 228–239.
21. Lee, K.H., Kay, J., Kang, B.H.: Keyword association network: a statistical multi-term indexing approach for document categorization. Proc. Fifth Australasian Document Computing Symposium, Brisbane, Australia, (2000) 9 - 16.
22. Lee, K., Kay, J., Kang, B.H.: KAN and RinSCut: lazy linear classifier and rank-in-score threshold in similarity-based text categorization. Proc. ICML-2002 Workshop on Text Learning, University of New South Wales, Sydney, Australia , 36-43 (2002)
23. Lee, K.H., Kay, J., Kang, B.H., Rosebrock, U.: A comparative study on statistical machine learning algorithms and thresholding strategies for automatic text categorization. Proc. PRICAI 2002, Tokyo, Japan, (2002) 444–453.
24. Lee, K.H., Kang, B.H.: A new framework for uncertainty sampling: exploiting uncertain and positive-certain examples in similarity-based text classification. Proc. Internat. Conf. on Information Technology: Coding and Computing (ITCC2004), Las Vegas, Nevada, 2004, 12pp.
25. Luger, G.F, “Artificial Intelligence. Structures and Strategies for Complex Problem Solving”. Addison-Wesley, 2005.
26. Mount, D.: “Bioinformatics: Sequence and Genome Analysis”. Cold Spring Harbor Laboratory, (2001). http://www.bioinformaticsonline.org/
27. Park, S.S., Kim, Y., Park, G., Kang, B.H., Compton, P.: Automated information mediator for HTML and XML Based Web information delivery service. Proc. 18th Australian Joint Conf. on Artificial Intelligence , Sydney, 2005, 401–404.
28. Park, G.S., Kim, Y.S., Kang, B.H.: Synamic mobile content adaptation according to various delivery contexts. J. Security Engineering 2 (2005) 202-208.
29. Park, G.S., Kim, Y.T., Kim, Y., Kang, B.H.: SOAP message processing performance enhancement by simplifying system architecture. J. Security Engineering 2 (2005) 163–170.

30. Park, G.S., Park, S., Kim, Y., Kang, B.H.: Intelligent web document classification using incrementally changing training data Set, J. Security Engineering 2 (2005) 186–191.
31. Păun, G. and Salomaa, A.: "New Trends in Formal Languages". Springer-Verlag, Berlin, 1997.
32. Petrovskiy, M.: Probability estimation in error correcting output coding framework using game theory. AI 2005: Advances in Artificial Intelligence, Sydney, Australia, 2005, Lect. Notes Artificial Intelligence **3809** (2005) 186–196.
33. Pin, J.E.: "Formal Properties of Finite Automata and Applications". Lect. Notes Computer Science **386**, Springer, New York, 1989.
34. Rozenberg, G. and Salomaa, A.: "Handbook of Formal Languages". Vol. 1, Word, Language, Grammar, Springer-Verlag, Berlin, 1997.
35. Smyth, B.: "Computing Patterns in Strings". Addison-Wesley, 2003.
36. Sugeng, K.A., Miller, M., Slamin and Bača, M.: (a, d)-edge-antimagic total labelings of caterpillars. Lecture Notes in Comput. Sci. **3330** (2005) 169–180.
37. Tuga, M. and Miller, M.: Δ-optimum exclusive sum labeling of certain graphs with radius one. Lecture Notes in Comput. Sci. **3330** (2005) 216–225.
38. van Leeuwen, J.: "Handbook of Theoretical Computer Science". Vol. A,B, Algorithms and Complexity. Elsevier, Amsterdam, 1990.
39. Witten, I.H. and Frank, E.: "Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations". Morgan Kaufmann, 2005.