Interception Method - Mathematical Substantiation
And Application In Columns-Based Intelligent Systems

Chesnokov A.M.
Cand. Sc., Institute of Control Sciences of RAS, Moscow.

Abstract. The article considers a variant of the intersection method proposed for solving basic problems in columns-based intelligent systems. In these systems, basic problems serve as the basis for solving all other problems. Basic concepts and definitions are given. The formulation of basic problems is described and their solution is given using a general universal method based on element-by-element comparison of patterns. Further, the solution of basic problems using the intersection method for patterns in the form of finite unordered sets and finite sequences or vectors is considered. The type of intersection used, the memorization operations are considered in detail, the necessary conditions are indicated, under which the correctness of the operation of the intersection method is proved. An estimate of the computational efficiency of the intersection method in comparison with the method based on element-by-element pattern comparison is given. In conclusion, the possibility of parallelizing calculations in solving basic problems, including basic problems for pattern regions, is shown.

Keywords: artificial intelligence, columns-based intelligent systems, column, intersection method, pattern region.

1. Introduction
The intersection method was originally proposed to solve pattern recognition problems. In columns-based intelligent systems, it is used as one of the possible methods for solving basic problems, which is more efficient than the method based on element-by-element pattern comparison. In these systems, basic problems serve as the basis for solving all other problems.

Column-based intelligent systems – these are universal systems that are considered within the framework of the following model [1, 2].

There is, albeit a very large, but finite set of names \( U \), intended for naming objects of an arbitrary nature. Without loss of generality, it is assumed that the set of names \( U \) is a finite subset of the set of integers \( Z \).

In the set of names, non-overlapping subsets are distinguished, which are called name domains. The number of allocated name domains is not constant. New name domains can be introduced at any time, and additional elements can be added to any name domain. The
allocation of name domains in real subject areas can be caused by various reasons (for example, typing). One of the most important reasons is the need to ensure that there are no accidental name matches in different parts of a large-scale system.

Any finite set of names belonging to one or another name domain is called a pattern.

The patterns of any set of patterns \( P \) can be renumbered using the names of a some name domain \( U_p \):

\[
P = \{ p_i \mid i \in U_p \}.
\]

An ordered pair \( (i, p) \) is called a column. The column is denoted as \( (i \mid p) \), where \( i \) is the column name, \( p \) is the column pattern. The notation \( i \rightarrow p \) is also used. In this case, the column name \( i \) is said to be a reference or pointer to a column pattern \( p \). In turn, it will be said about the pattern \( p \) itself that this pattern has a name \( i \) or is known by the name \( i \). (It should be noted that the element \( p \in P \), obviously corresponds to the pair \( (i, p) \) and column \( (i \mid p) \). The notation \( (i \mid p) \) is redundant and is used for convenience).

The mapping \( \varphi : i \rightarrow p \) is called name mapping.

Naming an pattern \( p \) with a name \( i \), or giving an pattern \( p \) a name \( i \), means that the definition of the name mapping \( \varphi \) an addition \( (i, p) \) is made such that, \( \varphi(i) = p \).

The name \( i \), which has not yet been used to name patterns is called a pure or empty name.

By default, the name mapping is considered to be one-to-one. All cases where this is not the case are considered separately.

So, if the same pattern \( p \) has several names \( i_k \), then the so-called pattern factorization takes place. To name mapping \( \varphi : i \rightarrow p \) all names \( i_k \) of the pattern \( p \) form an equivalence class \([i] = \{ i, \in U_p \mid \varphi(i) = p \} \). The one-to-one correspondence between the pattern name and the pattern is preserved, but in the form \( \varphi : [i] \rightarrow p \), where the name mapping \( \varphi : [i] \rightarrow p \) is defined as \( \varphi([i]) = \varphi(i_k) \) for \( \forall i_k \in [i] \). As a result, several columns \( i_k \rightarrow p \) can be replaced with a single column \( [i] \rightarrow p : \)

\[
p \ldots p \\
\uparrow \ldots \uparrow \Rightarrow \uparrow \\
i_1 \ldots i_n \quad [i] = \{i_1, \ldots, i_n\}
\]

If several patterns \( p_k \) have the same name \( i \), then factorization by name takes place. In this case, we consider the inverse mapping \( \varphi^{-1} : p \rightarrow i \). All patterns \( p_k \) form an equivalence class \([p] = \{ p, \in P \mid \varphi^{-1}(p_k) = i \} \). The one-to-one correspondence between the pattern and its name is preserved, but in the form \( \varphi^{-1} : [p] \rightarrow i \), where the inverse mapping \( \varphi^{-1} : [p] \rightarrow i \) is defined as \( \varphi^{-1}([p]) = \varphi^{-1}(p_k) \) for \( \forall p_k \in [p] \). As a result, several columns \( i \rightarrow p_k \) can be replaced by a single column \( i \rightarrow [p] : \)

7517 http://www.webology.org
Moreover, if each of these patterns \( p_k \) consists of only one name \( p_k = \{i_k\} \), then as a result of factorization by name, several columns \( i \rightarrow (i_k) \) can be replaced by one column \( i \rightarrow (i_1, \ldots, i_m) \), since \( [p] = \{i_1, \ldots, i_m\} \):

\[
\{i_1\} \ldots \{i_m\} \Rightarrow \{i_1, \ldots, i_m\}
\]

An index is any finite set of columns. The composition of any index can be changed by adding or removing columns. These operations are called addition and subtraction and are denoted by + and –. In the example below, to the index \( A \) is added \( l \) columns \( (i_k | p_k) \):

\[
A + \sum_{k=1}^{l} (i_k | p_k).
\]

If in the above example the patterns \( p_k \) are equal and consist of one name \( p_k = \{i\} \), then due to factorization by the pattern, the above sum can be written as:

\[
A + \sum_{k=1}^{l} (i_k | \{i\}) = A + (p | \{i\}),
\]

where the pattern \( p = \{i_1, \ldots, i_k\} \).

Obviously, the index can be represented as a table consisting of entries of the form "column name – names included in the column pattern". Such a table in vertical form consists of columns of variable height. In the bottom row of the table, under the line, are the names of the columns. Above the name of each column, all the names included in the column pattern are listed. By default, column names and names in patterns are considered to belong to different name domains.

If the patterns are unordered sets of names, then the order of the names in the column patterns can be arbitrary. Below, as the simplest example, there is an index \( A \) with patterns in the form of unordered sets, consisting of three columns \((1 | \{1, 3\}), (2 | \{2, 3, 4\}) \) and \((3 | \{4, 5\})\).

<table>
<thead>
<tr>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
<tr>
<td>3 3 4</td>
</tr>
<tr>
<td>1 2 5</td>
</tr>
<tr>
<td>1 2 3</td>
</tr>
</tbody>
</table>
If the patterns are ordered, then the names in the column patterns are written in a certain order, for example, from bottom to top, i.e. the first pattern name is in the first row above the line, the second one is in the second row, and so on. Such an order of notation is adopted in this article and in other works devoted to column-based intelligent systems.

A column-based intelligent system consists of one or more indexes and a mechanism that works with them, called a column engine. Receiving information about the outside world in the form of patterns, the column engine forms new columns, modifies existing ones, removes unnecessary ones, and performs all other operations with columns.

Knowledge in the systems under consideration is represented by columns, and the process of accumulation of knowledge is based on the memorization of new patterns under certain names. Obviously, the elementary basic problems, without the solution of which the functioning of such a system is impossible, are the direct problem (given a pattern, obtain its name) and the inverse problem (given a name, obtain the corresponding pattern). Memorization of new patterns is carried out as a part of the direct problem. If, when solving the direct problem, an unnamed pattern is found, then the column engine assigns a certain name to it and saves the corresponding data.

From a formal point of view, memorizing any pattern under a certain name always means the formation of the corresponding column \((i | p_i)\). At the same time, this does not mean that data will be stored in this form inside the system. The internal representation of the stored data is determined only by the method for solving basic problems and the way this method is implemented. The internal representation may differ significantly from the formal description in the form of a column \((i | p_i)\). An example of a method for which the formal description coincides with the internal representation of data is the method based on element-by-element pattern comparison. In other cases, the formed column \((i | p_i)\), will most likely be stored in an implicit form, and the solution of basic problems will not only show its existence, but also allow you to get its pattern by the column name, and its name by the pattern.

Solving basic problems, the column engine actually implements in the direct problem the transition from an pattern to a name by reference \(p_i \rightarrow i\) and in the inverse problem the transition from a name to an pattern by reference \(i \rightarrow p_i\). This provides the basis on which the solution of all other problems is built. The solution to any such problem is a sequence of references until the result is obtained.

Since everything is finite in the model under consideration, the solution of basic problems always exists. At the same time, the already mentioned method based on element-by-element pattern comparison is a general universal method for solving them. From the point of view of theory, this is enough to evaluate the possibilities of solving various problems using columns-based intelligent systems. However, if we talk about the practical application of such systems, then more efficient methods for solving basic problems, especially problems of large dimensions, are needed.

One of the possible methods for more efficient solution of basic problems is the intersection method. The idea of the intersection method goes back to the book index. In it, for each rubric, there are many pointers to those pages of the book where this rubric can be
found. A query from several headings obviously corresponds to the intersection of sets of pointers for these headings.

In the early 2000s A.M. Mikhailov showed that the intersection method can be used to pattern identification [7, 8]. Within the framework of the emerging direction, called the index approach, the intersection method is used mainly in solving problems of pattern recognition [9, 10].

Based on the results of [7, 8], the author proposed a variant of the intersection method intended for research in the field of columns-based intelligent systems [1, 2]. It was used to solve basic problems for patterns in the form of unordered finite sets, for patterns in the form of finite sequences or vectors, and also for patterns in the form of finite multisets [3]. It was used to show the possibility of implementation in columns-based intelligent systems of arbitrary functions \( f : U^n \rightarrow U^m \), including arbitrary Boolean functions \( f : B^n \rightarrow B^m \), where \( B = \{0, 1\} \) [4]. In addition, the possibility of implementing arbitrary relations (predicates) \( r \subset U^n \) in such systems was shown [5], as well as the possibility of solving basic problems under incomplete information, when for one reason or another only a part of the original pattern enters the system [6]. This article provides a mathematical description of the proposed version of the intersection method, which allows using this method not only in columns-based intelligent systems, but also for solving various problems of pattern recognition.

The next section describes the formulation of basic problems and their solution using a general universal method based on an element-by-element comparison of patterns. This serves as the basis for the subsequent comparison with the intersection method. Then we consider the solution of basic problems using the intersection method for patterns in the form of finite unordered sets and patterns in the form of finite sequences or vectors. The type of intersection used, memorization operations are discussed in detail, conditions are indicated under which the correctness of the intersection method is proved. Next, we estimate the computational efficiency of the intersection method in comparison with the method based on element-by-element pattern comparison. Finally, in conclusion, we show the possibility of parallelizing calculations in solving the direct problem both for patterns and for regions of patterns.

2. Basic problems and their solution using the element-by-element pattern comparison method

The direct and inverse problems are elementary basic problems, without the solution of which the functioning of columns-based intelligent systems is impossible. Solving basic problems, the column engine actually implements the transition from pattern to name by reference \( p_i \rightarrow i \) in the direct problem and the transition from name to pattern by reference \( i \rightarrow p_i \) in the inverse problem. This serves as the basis on which the solution of all other problems is built. The solution to any such problem is a sequence of references until the result is obtained.

The following statement of basic problems is considered.

Let \( P \) – be some non-empty set of patterns, and \( U_p \) – be a name domain for naming patterns \( p \in P \).
In order to solve the direct problem for any pattern $p \in P$, you need to get the name of the pattern $p$. If the pattern $p$ has not yet been remembered and is new, then it needs to be memorized under a certain name $i_p \in U_p$ so that the next time when solving the direct problem for the pattern $p$ the name $i_p$ is obtained.

In order to solve the inverse problem for any name $i \in U_p$, it is necessary to obtain an pattern $p$, known by the name $i$. If the given name $i$ is a pure name that has not yet been used for pattern naming, then an appropriate conclusion must be made.

A general universal method for solving basic problems is a method based on element-by-element pattern comparison. It is suitable in general for all types of patterns for which element-by-element comparison makes sense.

To solve basic problems using the element-by-element comparison method, it is sufficient to have one index $A$, which consists of columns $(i\mid p_i)$, where $p_i$ – is a pattern by name $i$.

Initially index $A = \emptyset$.

Direct problem solution. Let any pattern $p \in P$ be given, for which it is necessary to solve the direct problem. The pattern $p$ is compared element by element with the patterns of all columns of the index $A$. If a match of patterns is found, then the name $i$ of the column $(i\mid a_i) \in A$ is such that $p = a_i$, is the name of the pattern $p$ and is the solution to the direct problem.

If no match is found, then the pattern $p$ is new and needs to be memorized. The column engine chooses any pure name $i_p \in U_p$ for it and performs the addition

$$A + (i_p\mid p),$$

i.e. to the index $A$ a column $(i_p\mid p)$ is added. The name $i_p$ is the solution to the direct problem and is the name by which the pattern $p$ will now be known. If the direct problem is solved for the pattern $p$ again, then the name $i_p$ will be obtained.

Solution of the inverse problem. Let any name $i \in U_p$ be given. The pattern $p \in P$, known by the name $i$, is equal to the pattern $a_i$ of the column $(i\mid a_i) \in A$. If in the index $A$ there is no column with this name then $i$ is a pure name.

3. Solving basic problems using the intersection method for patterns in the form of finite unordered sets

Patterns are considered in the form of finite unordered sets of names $p = \{i_1, \ldots, i_m\}$, $m \geq 1$, where $i_k \in U_{(i)}$, $U_{(i)}$ – is the name domain for pattern elements. The number of elements of the pattern $p$ will be denoted by $m_p$, i.e. $m_p = |p|$, where $|S|$ – is the number of elements (cardinality) of the set $S$. 

http://www.webology.org
Obviously, the set of patterns under consideration is equal to the set $P_{(i)}$, which is the set of all subsets of the set $U_{(i)}$, excluding the empty subset.

To solve basic problems using the intersection method for patterns $p \in P_{(i)}$ we will use: an index $A$ for the direct problem, an index $B$ for the inverse problem, and also given by a set of ordered pairs $(i, m_i)$ function $m(i)$, which will store the number of elements of known patterns.

The formulation of the basic problems remains the same.

**Pattern storage operations**

In order to memorize the pattern $p = \{i_1, \ldots, i_m\} \in P_{(i)}$ under the name $i_p \in U_p$, the column engine performs the following additions:

$$A + (p \mid \{i_p\}) = A + \sum_{k=1}^{m} (i_k \mid \{i_p\}),$$

$$B + (i_p \mid p),$$

$$m(i) \cup (i_p, m_p),$$

i.e. to the index $A$ are added $m$ columns $(i_k \mid \{i_p\})$ for all names $i_k \in p$, to the index $B$ a column $(i_p \mid p)$ is added, and in the function definition $m(i)$ pair $(i_p, m_p)$ is added, where $m_p = m$.

Any name $i_k \in p$ can belong to several different patterns that the system memorized. Factorization by name causes the pattern $a_k$ of the column $(i_k \mid a_k) \in A$ to contain the names of all memorized patterns $p$ such that $i_k \in p$. This is the basis of the intersection method, since the intersection of patterns $a_{k_1}$ and $a_{k_2}$ columns $(i_{k_1} \mid a_{k_1}), (i_{k_2} \mid a_{k_2}) \in A$ will contain the names of all stored patterns $p$ such as $i_{k_1} \in p$ and $i_{k_2} \in p$ or $\{i_{k_1}, i_{k_2}\} \subset p$.

**Intersection and its properties**

For patterns $p = \{i_1, \ldots, i_m\} \in P_{(i)}$ the intersection will be used

$$\eta(A, p)_{(i)} = \prod_{k=1}^{m} a_k,$$

where $a_k$ – the pattern of the column $(i_k \mid a_k) \in A$, $i_k \in p$, $k = 1, \ldots, m$. Moreover, if in the index $A$ there is no column with name $i_k$, then it is believed that its pattern $a_k = \emptyset$.

Let the system memorize only new patterns and be given some pattern $p \in P_{(i)}$.

Obviously if $\eta(A, p)_{(i)} = \emptyset$, then the pattern $p$ is not memorized and is new. Otherwise, there would be columns for all $i_k \in p$, and the intersection $\eta(A, p)_{(i)} \neq \emptyset$, since it would contain the name $i_p$ of the pattern $p$. 

7522 http://www.webology.org
If \( \eta(A, p)_{(i)} \neq \emptyset \) and \( m(i) \neq m_p \) for all \( i \in \eta(A, p)_{(i)} \), then the pattern \( p \) — it is also a new pattern. Otherwise, the intersection \( \eta(A, p)_{(i)} \) would not only contain the name \( i_p \) of the pattern \( p \), but equality \( m(i_p) = m_p \) would also hold.

Finally, if \( \eta(A, p)_{(i)} \neq \emptyset \) and there is at least one name \( i \in \eta(A, p)_{(i)} \) such that \( m(i) = m_p \), then \( i \) is the name of the pattern \( p \), that it received when memorizing. Let us show that such a name is unique.

Suppose there are names \( i_{p1}, i_{p2} \in \eta(A, p)_{(i)} \) such that \( i_{p1} \neq i_{p2} \) and \( m(i_{p1}) = m(i_{p2}) = m_p \). Since the pattern \( p \) cannot receive two names at once, it first received the first name \( i_{p1} \), and then the second name \( i_{p2} \). Moreover name \( i_{p2} \) it would receive despite the fact that \( i_{p1} \in \eta(A, p)_{(i)} \) and \( m(i_{p1}) = m_p \). This contradicts the fact that only new patterns are memorized. Therefore, \( i_{p2} = i_{p1} \), i.e. if \( i_{p1} \neq i_{p2} \), then \( p_1 \neq p_2 \). Thus, if there is a name \( i_p \in \eta(A, p)_{(i)} \) such that \( m(i_p) = m_p \), then this name is unique.

**The necessary conditions**

For the intersection method to work correctly, the condition must be met, which is that different patterns must have different names, i.e. if \( p_i \neq p_j \), then \( i_{p1} \neq i_{p2} \).

Let us show that this condition is indeed a necessary condition.

Let two different patterns \( p_i \) and \( p_j \), for which all elements are different, have the same name \( i_p \) and equal number of elements \( m_{p1} = m_{p2} = m_p \). Consider a new pattern \( p_s \) with the same number of elements \( m_{p3} = m_p \). Let it have one part of the elements from the pattern \( p_i \), and another part of the elements from the pattern \( p_j \). Obviously, the pattern \( p_s \neq p_i \) and \( p_s \neq p_j \). Wherein \( i_p \in \eta(A, p)_{(i)} \) and \( m(i_p) = m_p = m_p \). Therefore, in this case, the intersection method will not be able to distinguish the pattern \( p_s \) from the patterns \( p_i \) and \( p_j \).

It is easy to see that the necessary condition is satisfied if a pure name \( i_p \in U_p \) is chosen for any new pattern, i.e. a name that has not yet been used for naming patterns \( p \in P_{(i)} \).

It should be emphasized that the necessary condition has practically no effect on the universality of the intersection method. So, for example, if different patterns \( p_1, \ldots, p_s \) need to be associated with the same common name \( i_p \), then for them the direct problem is first solved, subject to the necessary condition. Then an index is formed

\[
A_{(i)} = \{(i_{p1} \mid \{i_p\}), \ldots, (i_{pd} \mid \{i_p\})\},
\]

where \( i_{pk} \) — name of the pattern \( p_k \), which was obtained under the fulfillment of the necessary condition, \( i_p \) — common name of patterns \( p_1, \ldots, p_s \). Common name \( i_p \) for any pattern \( p_k \) can be obtained by taking for the name \( i_{pk} \) the pattern \( \{i_p\} \) of the column \((i_{pk} \mid \{i_p\}) \in A_{(i)}\), \( k = 1, \ldots, m \).
Solution of basic problems

The scheme for solving basic problems for patterns $p \in P_i$ has the form.

In the initial state $A = \emptyset$, $B = \emptyset$ and $m(i) = \emptyset$.

Direct problem solution. Let any pattern be given $p \in P_i$ with number of elements $m_p$.

For the pattern $p$ the intersection $\eta(A, p)_i$ is calculated.

If $\eta(A, p)_i \neq \emptyset$ and there is at least one name $i \in \eta(A, p)_i$ such that $m(i) = m_p$, then $i$ – this is the only name that represents the name of the pattern $p$ and is a solution to the direct problem.

In all other cases, the pattern $p$ is new and needs to be memorized. The column engine chooses for it any pure name $i_p \in U_p$ and does the additions

$$A + (p \mid i_p), \quad B + (i_p \mid p), \quad m(i) \cup (i_p, m_p).$$

The name $i_p$ is a solution to the direct problem and is the name by which the pattern $p$ will now be known.

Solution of the inverse problem. The inverse problem is solved in the same way as it was done in the element-by-element comparison method. For $\forall i \in U_p$ pattern known by name $i$, is equal to the pattern $b_i$ of the column $(i \mid b_i) \in B$. If in the index $B$ there is no column with this name, then $i$ – this is a pure name.

EXAMPLE. Let the indices $A$, $B$ and function $m(i)$ look like:

$$
\begin{array}{c|c|c|c|}
A & B & m(i) \\
\hline
3 & 3 & 3 & 3 \\
1 & 2 & 1 & 2 \\
3 & 2 & 4 & 1 \\
1 & 3 & 2 & 4 \\
\end{array}
\begin{array}{c|c|c|c|}
3 & 4 & 2 \\
1 & 2 & 1 \\
1 & 2 & 3 \\
\end{array}
$$

It is easy to see that the system has memorized three patterns: pattern $\{1, 3\}$ named 1, pattern $\{2, 4\}$ named 2 and pattern $\{1, 2, 3\}$ named 3.

Let it be necessary to solve the direct problem for the pattern $p = \{1, 2, 3\}$. Intersection $\eta(A, p)_i = \{3\}$ and $m(3) = 3 = m_p$. Therefore, the pattern $p$ – is the pattern named 3.

Let now the pattern $p = \{2, 3\}$. Intersection $\eta(A, p)_i = \{3\}$, but $m(3) = 3 \neq m_p$. Therefore, the pattern $p$ is new and needs to be memorized. The column engine chooses for it a pure name 4. Then additions are made $A + (p \mid \{4\}) = A + (2 \mid \{4\}) + (3 \mid \{4\})$ and $B + (4 \mid p)$, and in the function definition $m(i)$ pair $(4, 2)$ is added. As a result, we will have:
If for the pattern $p = \{2, 3\}$ solve the direct problem again, then $\eta(A, p)_{(i)} = \{3, 4\}.$ Wherein $m(3) = 3 \neq m_p$ and $m(4) = 2 = m_p.$ Consequently, $p$ – this is an pattern named 4.

Let now solve the inverse problem for the name $i = 3.$ The pattern $p$ is equal to the pattern $b_i$ of the column $(3|b_i) \in B,$ i.e. $p = \{1, 2, 3\}.$

4. Solving basic problems using the intersection method for patterns in the form of finite sequences or vectors

Let us now consider the direct and inverse problems for patterns in the form of finite sequences or vectors $p = (i_1, ..., i_m),$ $1 \leq m \leq n.$ We will assume that any pattern $p$ belongs to the set

$$P_{(i)} = \bigcup_{m=1}^{n} P^m, \ P^m = U_1 \times ... \times U_m,$$

where $U_k$ – name domain of the k-th coordinate. The dimension of a sequence or vector $p = (i_1, ..., i_m)$ will be denoted by $m_p,$ i.e. $m_p = m.$

The main difference of the intersection method for patterns $p \in P_{(i)}$ is the division of the index $A$ into $n$ parts $A = \{A_1, ..., A_n\},$ where $A_k$ – is the index of the k-th coordinate. Operations and conditions change accordingly.

To solve basic problems using the intersection method for patterns $p \in P_{(i)}$ the following will be used: an index $A = \{A_1, ..., A_n\}$ for the direct problem, an index $B$ for the inverse problem, as well as given by the set of ordered pairs $(i, m)$ function $m(i),$ which will store the dimensions of known patterns.

The wording of the basic problems does not change.

Pattern storage operations

To memorize the pattern $p = (i_1, ..., i_m) \in P_{(i)}$ under the name $i_p \in U_p$ the column engine performs additions:

$$A + (p|\{i_p\}) = \{A_1 + (i_1|\{i_p\}), ..., A_m + (i_m|\{i_p\})\},$$
$$B + (i_p|p),$$
$$m(i) \cup (i_p, m_p),$$
where \( m_p = m \).

Any name \( i_k \) can be the value of the k-th coordinate for several different patterns that the system memorized. Factorization by name results that the pattern \( a_k \) of the column \((i_k | a_k) \in A_k\) will contain the names of all memorized patterns \( p \) such that the k-th coordinate is equal to \( i_k \). This is the basis of the intersection method for patterns \( P_{(i)} \), since the intersection of patterns \( a_{k_1} \) and \( a_{k_2} \) columns \((i_{k_1} | a_{k_1}) \in A_{k_1} \) and \((i_{k_2} | a_{k_2}) \in A_{k_2} \) will contain the names of all memorized patterns \( p \) such that the \( k_1 \)-th coordinate is \( i_{k_1} \), and the \( k_2 \)-th coordinate is \( i_{k_2} \).

**Intersection and its properties**

For patterns \( p = (i_1, ..., i_m) \in P_{(i)} \) intersection will be used

\[
\eta(A, p)_{(i)} = \bigcap_{k=1}^{m} a_k ,
\]

where \( a_k \) – is the pattern of the column \((i_k | a_k) \in A_k\), \( i_k \) – name that is the k-th coordinate of the pattern \( p \). If the index \( A_k \) does not contain a column named \( i_k \), then its pattern is considered to be \( a_k = \emptyset \).

Let the system memorize only new patterns and be given some pattern \( p \in P_{(i)} \).

Obviously, if \( \eta(A, p)_{(i)} = \emptyset \), then \( p \) – new pattern.

If \( \eta(A, p)_{(i)} \neq \emptyset \) and \( m(i) \neq m_p \) for \( \forall i \in \eta(A, p)_{(i)} \), then the pattern \( p \) – it is also a new pattern.

But if \( \eta(A, p)_{(i)} \neq \emptyset \) and there is at least one name \( i \in \eta(A, p)_{(i)} \) such that \( m(i) = m_p \), then this is the name of the pattern \( p \), which it received when memorizing. The name \( i \) is the only one, since it is impossible to re-memorize a known pattern under a different name.

**The necessary conditions**

As before, a necessary condition for the correct operation of the intersection method is that each pattern receives a unique name, which is ensured by choosing a pure name for it. If this condition is violated, the method will not be able to distinguish between patterns.

Indeed, let the patterns \( p_1 \) and \( p_2 \), which received the same name \( i_p \), differ in all coordinates and have the same dimension \( m_p \). Let's consider a new pattern \( p_3 \) of the same dimension \( m_p \), in which one part of the coordinates is equal to the coordinates of the pattern \( p_1 \), and the second part of the coordinates is equal to the coordinates of the pattern \( p_2 \). As a result, the pattern \( p_3 \neq p_1 \) and \( p_3 \neq p_2 \). In this case, obviously, \( i_p \in \eta(A, p_3)_{(i)} \) and \( m(i_p) = m_p \), i.e., the intersection method will not be able to distinguish the pattern \( p_3 \) from the patterns of \( p_1 \) and \( p_2 \).
Solution of basic tasks

The scheme for solving basic problems for patterns \( p \in P_{(i)} \) has the form.

In the initial state \( A = \emptyset, B = \emptyset \) and \( m(i) = \emptyset \).

Direct problem solution. Let any pattern \( p \in P_{(i)} \) of dimension \( m_p \) be given. It calculates the intersection \( \eta(A, p)_{(i)} \).

If \( \eta(A, p)_{(i)} \neq \emptyset \) and there is at least one name \( i \in \eta(A, p)_{(i)} \) such that \( m(i) = m_p \), then \( i \) – the only name that represents the name of the pattern \( p \) and is a solution to the direct problem.

In all other cases \( p \) – new pattern that needs to be memorized. The column engine chooses any pure name \( i_p \in U_p \) for it and performs additions:

\[
A + (p \mid i_p), \quad B + (i_p \mid p), \quad m(i) \cup (i_p, m_p).
\]

The name \( i_p \) is the solution to the direct problem and is the name by which the pattern \( p \) will now be known.

Solution of the inverse problem. The inverse problem is solved in the usual way. For \( \forall i \in U_p \) the pattern \( p \), known by name \( i \), is equal to the pattern \( b_i \) of the column \( (i \mid b) \in B \). If in the index \( B \) there is no column with this name, then \( i \) – it is a pure name.

EXAMPLE. Let \( n = 3 \) and there are the following indices \( A, B \) and a function \( m(i) \):

\[
\begin{array}{ccc}
A_1 & A_2 & A_3 \\
4 & 3 & 3 \\
1 & 2 & 2 \\
1 & 2 & 3 \\
\end{array}
\]

\[
\begin{array}{ccc}
3 & 2 & 2 \\
3 & 1 & 1 \\
1 & 3 & 3 \\
1 & 2 & 3 \\
\end{array}
\]

\[
\begin{array}{c}
m(i) \\
\hline
1 & 2 & 3 & 4 \\
2 & 3 & 3 & 3 \\
\end{array}
\]

It is easy to see that the system memorized the pattern \( (1, 3) \) under the name 1, under the name 2 – pattern \( (3, 1, 3) \), under the name 3 – pattern \( (3, 1, 2) \), and under the name 4 – pattern \( (1, 2, 2) \).

Let the direct problem be solved for the pattern \( p = (3, 1, 2) \). The intersection \( \eta(A, p)_{(i)} = \{3\} \), while \( m(3) = 3 = m_p \). Therefore, the pattern \( p \) – is the pattern known by name 3.

If the pattern \( p = (1, 3) \) is given, then \( \eta(A, p)_{(i)} = \{1\} \) and \( m(1) = 2 = m_p \), i.e. in this case \( p \) – it is a pattern named 1.

http://www.webology.org
Let now the pattern \( p = (3, 1) \). The intersection \( \eta(A, p)_{(i)} = \{2, 3\} \), but \( m(2) = m(3) = 3 \neq m_p \). Therefore, \( p \) – this is a new pattern that needs to be memorized. Column engine chooses pure name 5 for it. Then additions are made \( A + (p \mid 5) = \{A_1 + (3 \mid 5), A_2 + (1 \mid 5)\} \), \( B + (5 \mid p) \) and \( m(i) \cup (5, 2) \). As a result, we will have:

\[
\begin{array}{ccc}
A_1 & A_2 & A_3 \\
5 & 5 & 4 \\
4 & 3 & 3 \\
1 & 2 & 2 \\
1 & 2 & 3 \\
\end{array}
\]

\[
\begin{array}{ccc}
B & m(i) \\
3 & 2 & 2 \\
3 & 1 & 1 \\
1 & 3 & 3 \\
1 & 2 & 3 \\
\end{array}
\]

If for the pattern \( p = (3, 1) \) solve the direct problem again, then the intersection \( \eta(A, p) = \{2, 3, 5\} \), and besides \( m(2) = m(3) = 3 \neq m_p \), and \( m(5) = 2 = m_p \). Therefore, \( p \) – is an pattern named 5.

Let now it is necessary to solve the inverse problem for the name 3. The pattern \( p \) by name 3 is equal to pattern \( b_i \) of the column \( (3 \mid b_i) \in B \), i.e. \( p = (3, 1, 2) \).

5. **Computational efficiency of the intersection method**

Let the direct problem be solved for patterns in the form of finite sequences or vectors of the same dimension \( n \), where all pattern coordinates have the same number of possible values \( |U_k| = m_{U_k}, k = 1, \ldots, n \).

Suppose the system has memorized \( l \) patterns. Let’s compare the number of operations that must be performed when solving the direct problem using the element-by-element comparison method and using the intersection method.

In element-by-element comparison, the patterns known to the system are stored in the index \( A \), which is a set of columns of the form \((i \mid p_i)\), where \( p_i \) – is the pattern known by the name \( i \). When solving the direct problem for some pattern \( p = (i_1, \ldots, i_n) \) its coordinate-wise comparison with the patterns \( a_i \) of all columns of the index \( A \) is performed. If a matching pattern \( a_i = p \) is found, then the column name \( (i \mid a_i) \in A \), is the name of the pattern \( p \) and solution of the direct problem. The maximum number of operations that must be performed \( O_e = nl \).

Let us now estimate the number of operations required to solve the direct problem using the intersection method and the index \( A = \{A_1, \ldots, A_n\} \). The pattern of each \( m_U \) columns of the index \( A_k \) \( (k=1, \ldots, n) \) contains on average \( l/m_U \) names. When calculating the intersection \( \eta(A, p)_{(i)} \) it is necessary to perform \( O_i = nl/m_U \) operations, i.e. \( m_U \) times less than in the
case of a coordinate comparison. When using the intersection method for patterns in the form of unordered sets \( p = \{i_1, ..., i_m\} \) the estimate \( O_\tau = n \ell / m_\tau \) is also true. However, in this case \( m_\tau = |U_{(\tau)}| \) – the number of different names involved in the formation of patterns \( p = \{i_1, ..., i_m\} \).

It should also be noted that the intersection method is well parallelized. Let’s show it on the example of implementation of the intersection method with the help of a function \( f_\eta(i, p) \), hat for an pattern \( p = \{i_1, ..., i_m\} \) counts the number of occurrences of any name \( i \in U_p \) in the column patterns of the index \( A \).

Let’s return to the original statement of the direct problem with patterns in the form of finite sequences or vectors \( p = \{i_1, ..., i_m\} \in P_{(\tau)} \), \( 1 \leq m \leq n \), which are named using the name domain \( U_p \).

Consider an indicator function, which for a given subset \( a \subset U_p \) and any name \( i \in U_p \) is equal to

\[
I_a(i) = \begin{cases} 
1, & i \in a, \\
0, & i \not\in a.
\end{cases}
\]

Let there be an index \( A = \{A_1, ..., A_m\} \). The function \( f_\eta(i, p) \) for any pattern \( p = \{i_1, ..., i_m\} \in P_{(\tau)} \) and any name \( i \in U_p \) is defined as follows:

\[
f_\eta(i, p) = \sum_{k=1}^{m} I_{a_k}(i),
\]

where \( a_k \) – is the pattern of the column \( (i_k | a_k) \in A_k \), \( i_k \) – name that is the k-th coordinate of the pattern \( p \), \( k = 1, ..., m \).

Obviously, \( 0 \leq f_\eta(i, p) \leq m \).

Denote by \( U' \subset U_p \) the set of names \( U' = \bigcup_{k=1}^{m} a_k \), where \( a_k \) – is the pattern of the column \( (i_k | a_k) \in A_k \), \( i_k \) – is the name that is the k-th coordinate of the pattern \( p \). It’s easy to see, \( \eta(A, p) \subset U' \).

As was shown, when solving the direct problem using the intersection method, for any pattern \( p = \{i_1, ..., i_m\} \in P_{(\tau)} \) of dimension \( m_p = m \) the intersection \( \eta(A, p)_{(\tau)} \) is calculated.

If the intersection \( \eta(A, p)_{(\tau)} \neq \emptyset \) and there is at least one name \( i \in \eta(A, p)_{(\tau)} \subset U' \) such that \( m(i) = m_p \), then \( i \) – the only name that is the name of the pattern \( p \) and is a solution to the direct problem. At the same time, \( i \in U' \) and the function \( f_\eta(i, p) = m_p \), \( m(i) = m_p \).

If \( \eta(A, p)_{(\tau)} \neq \emptyset \) and \( m(i) \neq m_p \) for \( \forall i \in \eta(A, p)_{(\tau)} \) or \( \eta(A, p)_{(\tau)} = \emptyset \), then the pattern \( p \) is new and must be memorized. Moreover, for \( \forall i \in U' \), either \( f_\eta(i, p) = m_p \) and \( m(i) \neq m_p \), or \( f_\eta(i, p) < m_p \).
Scheme for solving the direct problem for any pattern \( p = (i_1, ..., i_n) \in P \) using the function \( f_q(i, p) \) has the form.

The pattern \( a_i \) of the column \((i, |a_i|) \in A\) is taken, where \( i - \) name, which is the first coordinate of the pattern \( p = (i_1, ..., i_n) \).

When \( a_i \neq \emptyset \) for all names \( i \in a_i \) the function \( f_q(i, p) \) is calculated.

If there exists a name \( i \in a_i \) such that \( f_q(i, p) = m_p = m(i) \), then \( i \) - is the only name that is the name of the pattern \( p \) and is the solution to the direct problem.

In all other cases, including \( a_i = \emptyset \), the pattern \( p \) is new and must be memorized. The column engine performs normal pattern storage operations.

Obviously, function evaluation \( f_q(i, p) \) for all names \( i \in a_i \) can be done in parallel.

6. Pattern regions with the Hamming metric

With minor modifications, the circuit with a function \( f_q(i, p) \) can be used to solve the direct problem for pattern regions with the Hamming metric. The need for pattern regions arises, for example, in the presence of interference. In this case, the comparison of patterns is performed with an accuracy that is specified using some metric \( \rho(p', p) \). It is believed that if for an unknown pattern \( p \) and an pattern \( p_i \) named \( i \) is satisfied \( \rho(p, p_i) \leq \rho \), then the unknown pattern \( p \) is the pattern by name \( i \) (with precision \( \rho(p, p_i) \leq \rho \)). Therefore, only that pattern \( p \) is considered as a new one, or which there is no pattern \( p_i \) named \( i \) such that \( \rho(p, p_i) \leq \rho \).

In this case, obviously, if the system has memorized some pattern \( p_i \) named \( i \), then in fact the name \( i \) is given to all patterns of the pattern region \( \Delta_i = \{ p \in P | \rho(p_i, p) \leq r \} \). The name \( i \) can be interpreted as a region \( \Delta_i \) name. The corresponding name mapping \( \varphi_i : i \rightarrow \Delta_i \) is equal \( \varphi_i = \varphi_i \circ \varphi_p \), where \( \varphi_i : p \rightarrow \Delta_i \) - is a one-to-one correspondence between the pattern \( p \), and the region \( \Delta_i \) (for a given \( r \)), \( \varphi_p : i \rightarrow p_i \) - name mapping for patterns \( p \), mapping composition \( (f_i \circ f_p)(x) = f_i(f_p(x)) \).

Based on all that has been said, the formulation of the direct problem for the pattern regions is changed.

In order to solve the direct problem for any pattern \( p \), it is necessary to specify the names of all pattern regions \( \Delta \) such that \( p \in \Delta \). If such pattern regions do not exist, then the pattern \( p \) is new and it must be memorized under some name \( i_p \) so that a new pattern area \( \Delta \) by name \( i_p \) is created and \( p \in \Delta \).

Further, for simplicity, patterns \( p \) will be considered, of the same dimension \( m_p = n \), i.e. patterns \( p \in P^w = U_1 \times ... \times U_n \), where \( U_k \) - is the name domain of the \( k \)-th coordinate.

For pattern regions, the Hamming metric \( \rho_h(a, b) \) will be used, which for patterns \( a = (a_1, ..., a_n) \) and \( b = (b_1, ..., b_n) \) can be defined as follows:

\[
\rho_h(a, b) = \left[ \{ k \in [1, ..., n] | a_k \neq b_k \} \right],
\]
i.e. \( \rho_n(a, b) \) – the number of non-coinciding coordinates.

Pattern region \( \Delta_n(p, r) = \{ p' \in P^n | \rho_n(p, p') \leq r \} \), where \( r \) – integer, \( 0 < r < n \).

Let \( p_i \in P^n \) pattern known by name \( i \). It is easy to see that for any pattern \( p \in \Delta_n(p_i, r) \) the condition \( f_n(i, p) \geq n - r \) is performed. If the pattern \( p \not\in \Delta_n(p_i, r) \), then \( f_n(i, p) < n - r \). Therefore, any pattern \( p \) belongs to the pattern region \( \Delta_n(p_i, r) \) if and only if the condition \( f_n(i, p) \geq n - r \) is satisfied.

This allows membership \( p \in \Delta_n(p_i, r) \) to be checked, without having to calculate a metric \( \rho_n(p_i, p) \) for the pattern \( p \).

The scheme for solving the direct problem for pattern regions has the form.

Let any pattern be given \( p \in P^n \). For all names \( i \in U_p' \) the condition \( f_n(i, p) \geq n - r \) is checked and name set \( S_p = \{ i \in U_p' | f_n(i, p) \geq n - r \} \) is formed.

If the set of names \( S_p \neq \emptyset \), then it is a solution to the direct problem, since it contains the names \( i \) of all known pattern regions \( \Delta_n(p_i, r) \) such that \( p \in \Delta_n(p_i, r) \).

If \( S_p = \emptyset \), then such pattern regions do not exist. The pattern \( p \) is new and needs to be memorized. The column engine chooses any pure name \( i_p \) for it and performs the usual memorization operations. This means creating a new pattern region \( \Delta_n(p_i, r) \) named \( i_p \). If in the future for any pattern from the region \( \Delta_n(p_i, r) \) the direct problem will be solved, then the set of names \( S_p \) will contain at least the name \( i_p \).

Obviously, the condition check \( f_n(i, p) \geq n - r \) for all names \( i \in U_p' \) can be done in parallel.

7. Results

In columns-based intelligent systems, basic problems serve as the basis on which the solution of all other problems is built. Thus, the capabilities and efficiency of such systems depend on the capabilities and efficiency of the method for solving basic problems.

When solving a direct problem for any pattern \( p \in P \) you need to get its name. If the pattern has not yet been memorized and is new, then it must be remembered under a certain name \( i_p \) that later, when solving the direct problem, a name \( i_p \) will be obtained for it.

When solving the inverse problem for any name \( i \in U_p' \), it is necessary to obtain an pattern \( p \), known by the name \( i \). If the name \( i \) has not yet been used for naming patterns and is pure, an appropriate conclusion must be made.

A general universal method for solving basic problems is a method based on element-by-element pattern comparison. It is very simple and with it you can solve basic problems for any type of patterns, for which element-by-element comparison generally makes sense. With its help, you can easily assess the possibility of solving certain problems. However, from a practical point of view, more efficient methods for solving basic problems, especially problems of high dimension, are needed. One such method is the intersection method.
The paper provides a mathematical justification for a variant of the intersection method, which was proposed for use in columns-based intelligent systems. It is considered the solution of basic problems for patterns in the form of finite unordered sets \( p = (i_1, \ldots, i_m) \in P_{(i)} \), \( m \geq 1 \), and patterns in the form of finite sequences or vectors \( p = (i_1, \ldots, i_m) \in P_{(i)} \), \( 1 \leq m \leq n \).

Proved the correctness of the solution of basic problems. A necessary condition for this is that each pattern receives a unique name when memorized, which is ensured by choosing a pure name from the name domain \( U_p \). If the necessary condition is violated, the intersection method, generally speaking, will not be able to distinguish patterns. It is also shown that the necessary condition has practically no effect on the universality of the intersection method.

The solution of basic problems using the intersection method for patterns \( p \in P_{(i)} \) and \( p \in P_{(i)} \) has the same scheme and differs only in the intersection and addition operations used.

The general scheme for solving basic problems has the form.

In initial state \( A = \emptyset, B = \emptyset \) and \( m(i) = \emptyset \).

Direct problem solution. For any pattern \( p \), for which \( m_p \) – is the number of elements or dimension, the intersection \( \eta(A, p) \) is calculated.

If \( \eta(A, p) \neq \emptyset \) and there is at least one name \( i \in \eta(A, p) \) such that \( m(i) = m_p \), then \( i \) – is the only name that represents the name of the pattern \( p \) and is a solution to the direct problem.

In all other cases, the pattern \( p \) is new and must be memorized. The column engine chooses any pure name \( i_p \in U_p \) and perform additions \( A + (p \mid \{i_p\}) \), \( B + (i_p \mid p) \), \( m(i) \cup (i_p, m_p) \).

The name \( i_p \) is a solution to the direct problem and is the name by which the pattern \( p \) will now be known.

Solution of the inverse problem. For any name \( i \in U_p \) pattern known by name \( i \), equal to the pattern \( b_i \) of the column \((i \mid b_i) \in B \). If in the index \( B \) there is no column with this name, then \( i \) – is a pure name.

The paper presents an assessment of the effectiveness of the intersection method in comparison with the method based on element-by-element comparison of patterns.

Let the direct problem for patterns be solved \( p = (i_1, \ldots, i_k) \in P_{(i)} \) of the same dimension \( m_p = n \), and all pattern coordinates have the same number of possible values \( |U_k| = m_u \), \( k = 1, \ldots, n \). If the system remembers \( l \) patterns, then the maximum number of operations that must be performed when solving the direct problem using the element-by-element comparison method is equal to \( O_e = ml \). When using the intersection method, it is necessary to perform \( O_i = nl / m_u \) operations, i.e. in \( m_u \) times less. A similar estimate is also valid for patterns in the form of finite unordered sets \( p \in P_{(i)} \), \( m_p = n \). In this case \( m_u \) – the number of different names involved in the formation of patterns \( p = (i_1, \ldots, i_k) \in P_{(i)} \).
The intersection method is well parallelized. This is illustrated by the implementation of the intersection method using a function \( f_\eta(i, p) \), that, for an pattern \( p \) counts the number of occurrences of any name \( i \in U_p \) in the corresponding index columns.

The scheme for solving the direct problem remains the same. However, instead of an intersection \( \eta(A, p)_i \), a function \( f_\eta(i, p) \) is calculated for all names \( i \) of the pattern \( a_i \neq \emptyset \) of the column \( (i_i | a_i) \in A_i \), where \( i_i \) is the name that is the first coordinate of the pattern \( p = (i_i, ..., i_n) \).

If a name \( i \in a_i \) is found, for which \( f_\eta(i, p) = m_p = m(i) \), then \( i \) is the only name that represents the name of the pattern \( p \), and is a solution to the direct problem.

In all other cases, including \( a_i = \emptyset \), the pattern \( p \) is new and needs to be memorized. The column engine performs normal memorization operations.

Obviously, function evaluation \( f_\eta(i, p) \) for all names \( i \in a_i \) can be done in parallel.

With minor modifications, the scheme with a function \( f_\eta(i, p) \) can be used to solve the direct problem for pattern regions with the Hamming metric \( \rho_H(p, p') \) (the number of non-coinciding coordinates).

For pattern regions, the formulation of the direct problem changes – when solving it for any pattern \( p \) it is necessary to indicate the names of all pattern regions \( \Delta \) such that \( p \in \Delta \). If such pattern regions do not exist, then by storing the pattern \( p \) under the name \( i_p \), a new pattern region \( \Delta \) named \( i_p \) is created.

For patterns \( p = (i, ..., i_n) \in P^n \) and metrics \( \rho_H(p, p') \) the region of patterns is understood as a set of patterns \( \Delta_H(p, r) = \{p' \in P^n | \rho_H(p, p') \leq r\} \), where \( r \) integer, \( 0 < r < n \). It was shown that if \( p \) is a pattern by name \( i \), then any pattern \( p \) belongs to the region \( \Delta_H(p, r) \) if and only if the condition \( f_\eta(i, p) \geq n - r \) is satisfied. This allows you to check the ownership \( p \in \Delta_H(p, r) \), without calculating for the pattern \( p \) the metric \( \rho_H(p, p) \).

The scheme for solving the direct problem for pattern regions has the form.

Let any pattern be given \( p \in P^n \). For all names \( i \in U_p \), the condition \( f_\eta(i, p) \geq n - r \) is checked and a name set \( S_p = \{i \in U_p | f_\eta(i, p) \geq n - r\} \) is formed.

If the set \( S_p \neq \emptyset \), then it is a solution to the direct problem, since it contains the names \( i \) of all known pattern regions \( \Delta_H(p, r) \) such that \( p \in \Delta_H(p, r) \).

If \( S_p = \emptyset \), then such pattern regions do not exist. The pattern \( p \) is new and needs to be memorized. The column engine chooses any pure name \( i_p \) and performs the usual pattern storage operations. This means creating a new pattern region \( \Delta_H(p, r) \) named \( i_p \). If in the future the direct problem is solved for any pattern from the region \( \Delta_H(p, r) \) then the name set \( S_p \) will contain at least the name \( i_p \).

Obviously, function \( f_\eta(i, p) \) evaluation and condition \( f_\eta(i, p) \geq n - r \) check for all names \( i \in U_p \) can be done in parallel.
8. Conclusion

Columns-based intelligent systems have great potential. Arbitrary functions and relations (predicates) can be implemented in them, they can solve problems of a classification type, work under conditions of incomplete information and noise, and perform logical inference. This is based on the basic problems. The variant of the intersection method proposed for their solution is quite simple, efficient and well parallelized. It can be used not only in columns-based intelligent systems, but also for solving pattern recognition problems.

References