19th Int. Workshop on IFSs, Burgas, 4–6 June 2015
Notes on Intuitionistic Fuzzy Sets
ISSN 1310–4926
Vol. 21, 2015, No. 2, 77–86

Software for InterCriteria Analysis: Implementation of the main algorithm

Deyan Mavrov

'Prof. Dr. Asen Zlatarov' University
1 Prof. Yakim Yakimov Blvd., 8010 Burgas, Bulgaria
e-mail: dg@mavrov.eu

Abstract: The InterCriteria Decision Making approach provides new ways to compare any two criteria, against which a set of objects have been evaluated. In order to test how this approach performs with different sets of data, a specialized software application has been developed, which takes two matrices of input data and outputs the intuitionistic fuzzy pairs that describe the intercriterial relationship as two tables. The application can work with Microsoft Excel workbooks or text files and provides ways to transfer the output data to other programs. It can also include functionality to display graphics of the output data.
Keywords: InterCriteria Decision Making, Index Matrix, Intuitionistic Fuzzy Pair, application software, C++, Qt.
AMS Classification: 03E72.

1 Introduction

The InterCriteria Analysis method described in [1] offers an interesting new way for comparison of the individual criteria among a set of criteria, against which a set of objects have been evaluated. This has been designed as a completely data driven method, which requires real data to practically see effect of its application. A specialized software application has been developed, which requires as input one two-dimensional array of data of the evaluation of the set of m objects against the set of n criteria, and after processing returns as output two n×n tables, the first of which contains the membership parts, and the second – the non-membership parts of the Intuitionistic fuzzy pairs that defined the degrees of correlation between any two criteria in the set of criteria.

In the sections below is given the principle of action of the software application, as well as some of its additional functionalities.

2 Programming language and additional libraries

The application is developed using the C++ language. The main platform for the application development is Microsoft Windows. The compiler used is the standard CL.exe program from Visual C++ 2012, without employing the Visual Studio graphic environment.

Based on previous experience, the Qt library was chosen for the application's graphical interface. Qt offers classes, which help building the on-screen graphic objects, as well as classes for non-graphic objects like (strings, database connections, etc.) [2]. Each object of a Qt class can interact with the rest of the program's objects, using a system of signals and slots, where a signal sent from one object can be connected to another object's slot. The Qt library is cross-platform, and works with a variety of operating systems and compilers, and in case only the standard Qt classes are used, an application designed under Windows, can be recompiled under Linux or Mac OS X with almost no changes.

Moreover, the Qt project includes a C++ development environment, Qt Creator. It offers a visual editor for design of windows, which significantly facilitates the use of the graphic interface.

3 Input data format

The choice of the data input method in an application always raises a lot of questions that need to be answered in order to make the interaction with that application as convinient and problem-free as possible. The most impractical method would be to have the data as part of the source code of the program, which would be handy in the testing period, but not for multiple executions with various datasets. For this reason, better alternatives are to be explored.

One of the alternatives is to use the well-known 'copy-paste' principle from graphic applications, in which the necessary information is copied from another application and pasted into the developed one. In the present case, this method would work well enough, but the 'copy' operation in most cases takes up time (mainly for selecting the data), especially if we want to execute the application with various input tables. Furthermore, during the 'copy' operation not all existing applications format the data in the same way. Data are also stored in the temporary memory (clipboard) in a raw form, which requires data separation at import.

Another option is to have the data stored in an external file, where they are ordered in a way that our application 'understands', and have this file opened prior to the algorithm's execution, in order to retrieve the data. Since in our case, the InterCriteria Analysis application will expect to obtain data in Microsoft Excel compatible formats, and copying the data into Excel and their consecutive reorganization again there, is usually fairly easy, hence, this is an adequate option.

For the reasons, listed above, for the ICA application it was chosen to have the data imported from a file, which is in one of the Microsoft Excel supported file formats. After this, it is necessary to determine the precise way of accessing these files. There are a number of libraries for operating with Excel files, some of which are not intended for use with C++. Since certain precision is aimed for the development of the present application, it is preferable for the file to maintain full compatibility with Excel. For this aim, the Windows supported

Components Object Model (COM) is used, which provides for some applications, including Excel, to be used as processing servers [3].

In the system header files of Visual C++ functions are provided for interaction with the COM server. In order to guarantee the bigger compatibility with its working principles, the Qt library provides the embedded system ActiveQt, which serves as a bridge between Qt and COM. One of the major advantages of ActiveQtis the easier retrieval of data as objects of the embedded classes of Qt, which, on their turn, contain a lot of functions for further modification. When working with ActiveQt, it is oftentimes used the class QVariant, which serves as a variable type, from which data in other types are additionally retrieved. On the other side Excel provides COM functions for most of the actions, which could be manually done in the graphic interface, as well as some specific actions, necessary for the operation of the COM server.

4 Main module of the program

4.1 Data reading

4.1.1 Using Microsoft Excel

After the application is launched, a window with empty fields and inactive buttons appears.

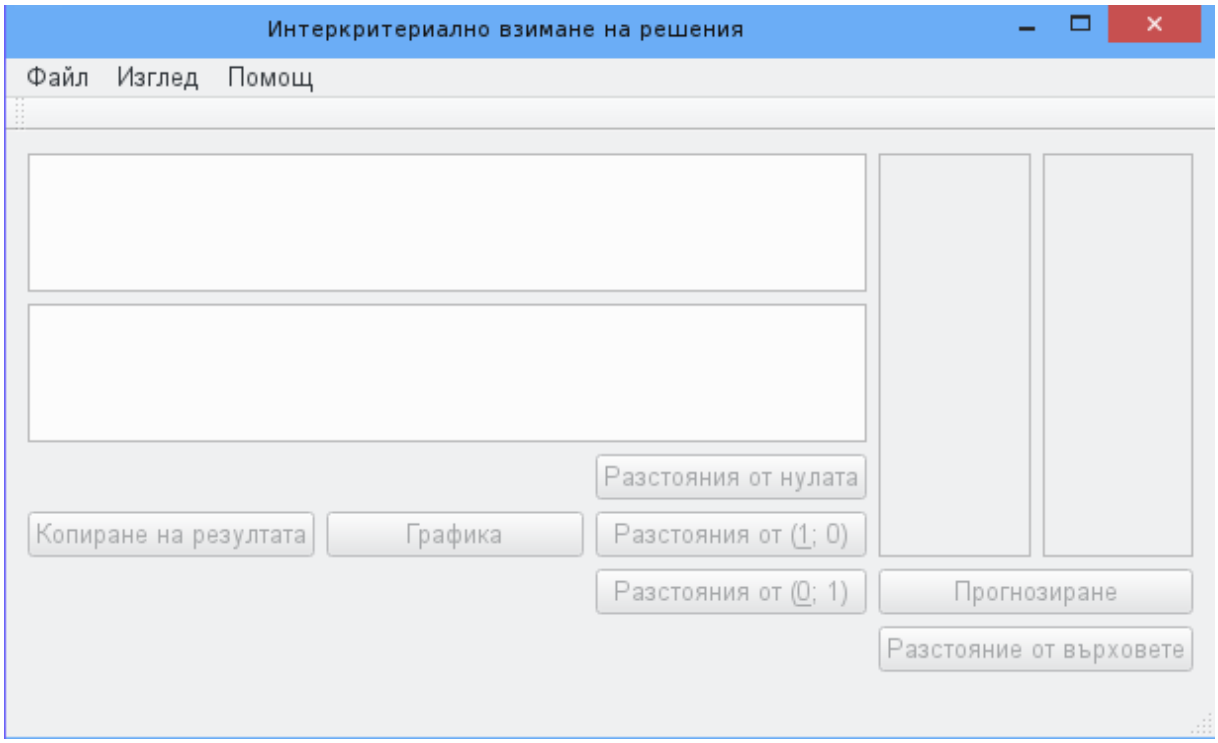

Figure 1. Main window at start-up, with no input data provided

Before the application can start its work, the first thing we need to provide is the name of the file that contains input data, which includes the evaluations of a set of objects against a set of criteria. This happens by selecting the option 'Open' from the 'File' menu. The application can also obtain the file name as a command-line parameter, and if no parameter has been provided, the 'Open File' dialog must be used. In the newer editions of MS Windows (and possibly other OS) it is possible to preview the file in this dialogue window before opening it.

In order for the application to access and read the file, it has to be from a file type supported in MS Excel (see above) and the data needs to be placed in the first worksheet. The input data consists of two index matrices (see [4]) where each matrix has rows of criteria evaluating columns of objects. The matrices need to be transferred in the first sheet, following strictly the ordering scheme shown in the table below:

	A	B	C	…	*z-1*	***z***
1	*Not used*	Name of object 1	Name of object 2	…	Name of object [z-2]	
2	Name of criterion 1.1	Evaluation C1.1-O1	Evaluation C1.1-O2	…	Evaluation C1.1-O[z-2]	
3	Name of criterion 1.2	Evaluation C1.2-O1	Evaluation C1.2-O2	…	Evaluation C1.2-O[z-2]	
…	…	…	…	…	…	…
a-1	Name of criterion 1.[a-2]	Evaluation C1.[a-2]-O1	Evaluation C1.[a-2]-O1	…	Evaluation C1.[a-2]-O[z-2]	
a				…		
a+1	Name of criterion 2.1	Evaluation C2.1-O1	Evaluation C2.1-O2	…	Evaluation C2.1-O[z-2]	
a+2	Name of criterion 2.2	Evaluation C2.2-O1	Evaluation C2.2-O2	…	Evaluation C2.2-O[z-2]	
…	…	…	…	…	…	…
x-1	Name of criterion 2.[x-1-a]	Evaluation C2.[x-1-a]-O1	Evaluation C2.[x-1-a]-O2	…	Evaluation C2.[x-1-a]-O[z-2]	
x				…		

Table 1. Standard ordering of the data in the input table

These are two tables, separated with an empty row. The cells under the second table, and to the right of both tables must also remain empty. The idea is to juxtapose the criteria from the first table with the criteria from the second table – the pairs of criteria comprise one criterion from the first table, and one from the second table. The objects that are being evaluated in both tables, have to fully coincide. If the table below is a copy of the table above (without the row for objects' labels), then the criteria are practically compared with each other, in order to discover dependencies between the set of criteria.

Any text formatting is ignored. Currently, for correct recognition and processing, the software requires that the evaluations are in numerical format. Missing and incomplete data raise two possibilities: 1. Removal of either the row, or column, to which the missing value belongs, with priority being given to keeping the column (criterion) rather than to the row (object) or 2. Keeping the row/column while increasing the uncertainty counter, as described below. If the specific problem allows or requires so, it is possible to transpose the objects/criteria table and thus search for correlations among the objects, rather than among the criteria.

After opening the file, the software starts reading the data. For this purpose, it calls the function OpenWorkbook. In the beginning it opens a connection to Excel as a COM server, opens the file through that connection, which creates an object, pointing at it. From this object, an object pointing at the first worksheet, is created. The latter gives access to functions for reading the cells' contents, some of which are later used in the application.

Now we will describe the entire process step by step.

First, the labels of the criteria are being read. For each of the two tables, an array from type vector is created (which for the first output table is named *cTitles*, and for the second output table is named *dTitles*). With respect to Table 1, this means the following:

- Starting with cell A2, the contents of each cell down the column is added to the array of labels with criteria from the first table (*cTitles*), until empty cell has been reached (in Table 1, this is cell A*a*);
- Starting with the first cell under the reached empty cell from the previous step, (In Figure 1, this is cell A[*a+1*]), the contents of each cell down the column is added to the array of labels with criteria from the second table (*dTitles*), until empty cell has been reached (in Figure 1, this is cell A*x*).

Then, it is necessary to read all cells with numerical values from both tables. For this purpose, in the classMainWindow, two arrays of arrays of decimal fractions (type *double*) are being defined (*vector<vector<double>>*) named, respectively *data1* and *data2*. Similarly to the reading of the criteria's labels, reading the rows from the first table starts with cell B2 and goes on until the first empty cell in a row (in Figure 1, this is cell Ba). For each line, reading continues to the right, until an empty cell has been reached (in Figure 1 – a cell from column z). The number of full cells on the first read row serves as the norm. If the number of cells in any of the following rows, whether in the first, or the second table, is different from the norm, the program execution terminates with an error dialog. After reaching the end of the first table, the processing of the second table is done likewise. Each row is aggregated in an array from type *vector<double>*, and after the end of reading that row, this array is added to the array of rows for the respective table (*data1* or *data2*).

The principle of reading (accessing) the rows of data consists in reading from the first cell until an empty cell has been reached. First we open a cell object using the worksheet object's *Cells(int, int)* function for the first numerical cell of the row. In this function, the first parameter is the row number, and the second parameter is the column number (counting starts from 1, rather than 0, as usual in C++). From this object we use the End(int) function to acquire an object for the row's last non-empty cell. The reading of each row is performed using the Excel COM function *Range(address1:address2)*, where *address1* and *address2* are respectively the start and end cell of the row. Of course, *Cells(int, int)* could be used for this as well, however reading cell by cell in previous versions of this application has turned out to work much slower than reading an entire range.

After completing these operations, the connection with Excel's COM server is interrupted and its process is closed.

If the currently open file is closed and another is opened, all hitherto described values are deleted and a new reading operation starts from the scratch. For manual deletion, an option in the 'View' menu of the program can also be used.

4.1.2 Tab-Delimited Files

The method described above can work only under the Windows operating system, whereas it would be useful if the described application can be executed under different platforms. Therefore, an alternative format is needed, that would cause less problems under OS such as Linux, and that would preferably need no further external library than Qt. This makes a text file format the perfect candidate for the task.

The way the data is ordered in these text files should follow the above described format to keep clarity and compatibility with previously prepared input files. In text files data rows are usually stored in single lines of text, however there are multiple methods to separate the single elements in these rows. The most popular is CSV (Comma-Separated Values), which however does not always look the same. Excel, for instance, saves CSV files with either commas or semicolons as a separator, depending on the system locale setting. Also, if a text cell contains commas or semicolons, Excel (and most similar spreadsheet software) surrounds the text with double quotes to distinguish it from the other data. These inconsistencies suggest that either special care must be taken when writing the reader function for such files, or that a helper library should be used.

Therefore, as the first text file format to be supported in this application, Tab-Delimited Files have been chosen. Tab-Delimited Files use tabulations as separators, which do not interfere with neither the dot nor the comma as decimal markers. Reading is done in a way similar to that used with Excel, though the row ends are here determined by the line end, and not by looking for an empty cell.

While Excel converts decimal fractions automatically to usable numbers for C++, with text files this conversion needs to be done using Qt. In order for this to work, the number's decimal point must be a dot (as the C++ standard specifies), so if the system locale has set it to a comma, the application replaces commas to dots for numeric cell data.

To make the application OS-independent, the Excel reading operations and the Tab-Delimited File operations are split to their own functions, and the Excel reading function is placed in an `#ifdef _WIN32 ... #endif` block. This also makes it easier to add further reading functions, for example for CSV or ODS (OpenDocument Spreadsheet) formats.

4.2 Calculation of the IF values

After the data in both tables have been checked and made available in the respective arrays, the program can start calculating the values of intercriteria correlation, i.e. the membership and non-membership that form intuitionistic fuzzy pairs (as described in [5]). The values for μ and ν for each pair of criteria are to be stored in two two-dimensional arrays, named respectively Ro and Sigma, which are of the type vector<vector<double>>. To facilitate this process, the following preparations have been made:

- The comparison function *compare* has been declared, which accepts two numbers as parameters. Depending on the sign, resulting from the comparison, the number *1*, *2* or *3* is returned, respectively for '<', '=' and '>'.
- Two arrays: *sum_m* and *sum_n* have been created with the same dimensions as *Ro* and *Sigma*, from type *vector<vector<int>>*. They store counters for the intercriterial sums – *sum_m* for the times in which the signs match, and *sum_n* for the times the signs contradict each other, when comparing the evaluations with two criteria of every two objects.

Afterwards, we can start accumulating the two sums in *sum_m* and *sum_n*. For each pair of objects, the pair of criteria numbered respectively *i* and *j* are taken, the first criterion from the first table, and the second one – from the second table. The function *compare* checks the sign between the evaluations of both objects against the first criterion. Then, again with compare, the sign is checked between the evaluations of the two objects against the second criterion. If the two signs are the same, then the counter in *sum_m[i][j]* increments with one. If the signs do not coincide, then the *sum_n[i][j]* counter increments with one. If the signs are different, and are one of them is '=', then none of these two counters increments, which practically means that the uncertainty factor is increased.

In order to accelerate this part of the program's operation, in subsequent versions of the application the results from the comparisons between pairs of numbers with function compare are calculated in advance for each given pair of columns from the two input matrices. They are stored in the arrays *x_values* (for the comparisons of the numbers from *data1*) and *y_values* (for the comparisons from *data2*), where the position of a given result of comparison is determined by the number of the row, from which both numbers are retrieved.

Since the checks for each pair of columns *k1* and *k2* are independent, it is possible to have each pair of columns detached in a separate thread of the process, in which the application is being executed. This is mostly useful in multi-core processor systems, and allows for each CPU core to process a separate pair of columns, in parallel with the other working CPU cores. This, however, calls for some structural changes in the program source, in order to keep the values of the counters *sum_m[i][j]* and *sum_n[i][j]* across the various threads. The class Counter needs to be defined, which contain the methods increment (for incrementing the counter), decrement (for decrementing the counter), reset (for resetting it to 0) andvalue (for retrieving its current value). These methods perform one operation each, but before the operation, a mutex semaphore temporarily blocks the given counter object, so that the separate increment methods do not interfere with one another, which otherwise may lead to a wrong value of the counter. The types of the two counter arrays are changed to *Counter* and, respectively, the calls to them in the application are rewritten in a way to use the above described methods.

The application allocates so many process threads, as is the number of available cores of the CPU (which in many configurations could be more than one). The threads are stored in an array of pointers, and the calculations for every subsequent pair of columns are directed subsequently to the next thread in the array.

The operations used for comparison and decisions which counter will increment when, are detached in a separate class, called *Calculations*, and, more precisely, in its method *doCalculate*. For each pair of columns, a separate object from this class is created. The main

class of the main window, called *MainWindow*, transfers the data from the two input matrices, using pointers to data1 and data2. In the same we obtain the pointers to the variables of the two counters, *sum_m[i][j]* and *sum_n[i][j]*. Once pointers to the necessary data have been directed to the object for calculations, as well as the index numbers of the compared columns from the input matrices, the object is transferred to one of the available process threads.

In order to avoid a situation of data loss, the applications waits for the number of processed pairs of columns to reach the total number. Only once this number has been reached, the application proceeds to the consequent step.

After all the sums have been accumulated, the values of μ and ν need to be calculated for each pair of criteria in the first and second tables. If N is the number of objects, this happens in the following way for each pair of criteria, labelled i and j (i from the first table, and j from the second one):

$$Ro[i][j] = \frac{sum\_m[i][j]}{\frac{N(N-1)}{2}}$$

$$Sigma[i][j] = \frac{sum\_n[i][j]}{\frac{N(N-1)}{2}}$$

4.3 Displaying the results

After the values in the output matrices are ready, the data from them have to be printed on screen. For this purpose, two objects from the class *QTableWidget* (which allows displaying data in a table) are placed in the main window, one below the other. The first table lists the degrees of membership between criteria, while the second lists the degrees of non-membership. Criterion names are printed in the respective column and row header boxes, which surround *QTableWidgets* by default. After completing the two tableobjects in the main window, they are printed onscreen. Assisting the userto better operate with the output data, a button 'Copy the result' is provided under the main window. Clicking it takes the content of both intercriteria tables in an Excel-compatible format, and stores it in the clipboard. From there, this content can be simply pasted in an Excel sheet.

In order for the copied data to be compatible with Excel and their programs, each separate element of the result needs to be separated from the others with a tabulation, and table rows need to be separated by newlines. The decimal marker depends on the system locale settings. Similarly to how the calculation results are displayed in the main window, the copied data is presented as two tables, one below the other (for membership and non-membership values, respectively). On each table's first row, in the leftmost cell, is printed the table's label (ρ for the membership pairs of the pairs and σ for the non-membership pairs), after which the names of the criteria from the second table are listed. On every subsequent row from the two tables, the first cell contains the name of the respective criterion i from the first input table, and in the next cells are listed the values from the respective matrix of the intuitionistic fuzzy values between criterion i and the current criterion j from the second table, whose column has been reached at this point.

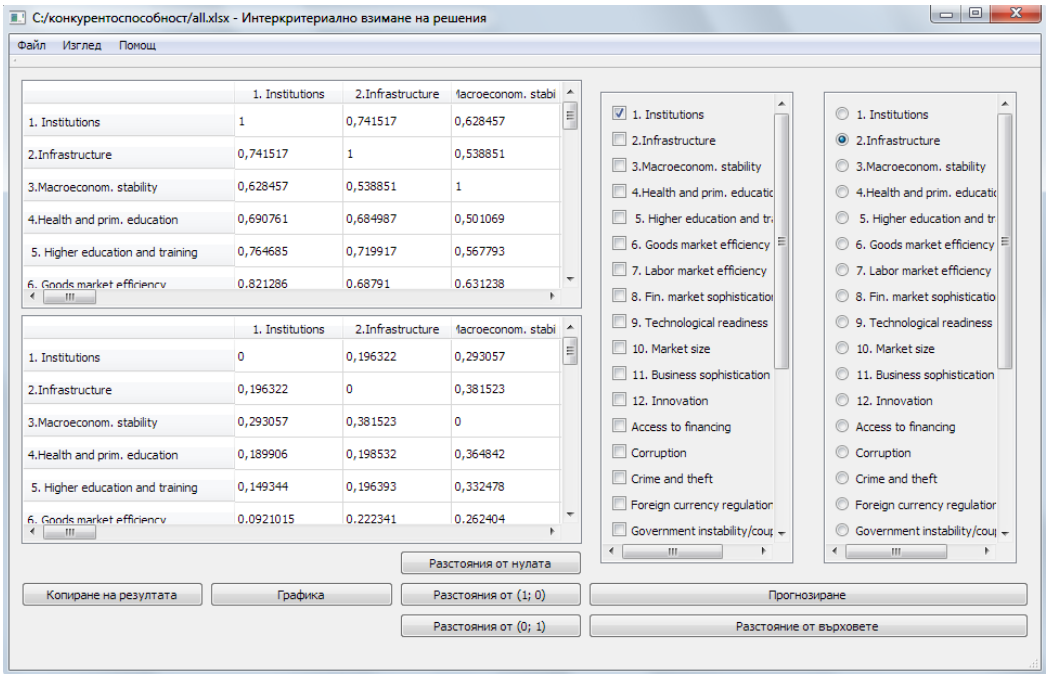

Figure 2. Main application window (March 2015).

The application's newer versions also provide tools to display the resulting data in the Intuitionistic Fuzzy interpretation triangle (see [5, 6]), as explained in [7].

5 Conclusions

The aim of this paper is description of the application for executing of the InterCriteria Analysis algorithms, and testing it over a wide variety of values, in order to evaluate its performance and results. The application has been so far tested with various data, among which: petrochemical data, data for EU Member States' competitiveness as taken from the World Economic Forum's Global Competitiveness Reports, and others (for further details see [8] [9] [10]).

A matter of further implementation and improvement are the ICA-based methods for data prognostics (missing data imputation), which will later be included in the respective module of the ICA software application.

If necessary, other functionalities are also possible, among which selection of consonance thresholds, in order to sieve the criteria that are in highest positive consonance with each other. In the present form, the application can be successfully used for analysis of new data, for detecting known correlations, and discovery of new, previously unknown correlations and knowledge.

Acknowledgements

The present research is supported by the National Science Fund of Bulgaria under Grant Ref. No. DFNI-I-02-5/2014 "InterCriteria Analysis – A New Approach to Decision Making".

References

[1] Atanassov K, Mavrov, D., & Atanassova, V. (2014) InterCriteria decision making. A new approach for multicriteria decision making, based on index matrices and intuitionistic fuzzy sets. *Issues in Intuitionistic Fuzzy Sets and Generalized Nets*, 11, 1–8.

[2] *Qt 5.4 Documentation: All Classes*. (2015) The Qt Company Ltd. Retrieved on March 15, 2015 from `http://doc.qt.io/qt-5/classes.html`

[3] Office Automation Using Visual C++. (2015) Microsoft Knowledge Base. Retrieved on March 15, 2015 from `https://support.microsoft.com/en-us/kb/196776`

[4] Atanassov, K. (2014) *Index Matrices: Towards an Augmented Matrix Calculus.* Springer, Cham.

[5] Atanassov, K. (1999) *Intuitionistic Fuzzy Sets: Theory and Applications*. Springer Physica-Verlag, Heidelberg.

[6] Atanassov, K. (2012) *On Intuitionistic Fuzzy Sets Theory*. Springer, Berlin.

[7] Mavrov, D., Radeva, I., Capkovic, F., & Doukovska, L. (2015) Software for InterCriteria Analysis: Graphic Interpretation within the Intuitionistic Fuzzy Triangle. *Proc. of 5th Int. Symp. on Business Modeling and Software Design*, 6–8 July 2015, Milano (accepted).

[8] Atanassova, V., Doukovska, L., Atanassov, K., & Mavrov, D. (2014) InterCriteria Decision Making Approach to EU Member States Competitiveness Analysis. *Proc. of 4th Int. Symp. on Business Modeling and Software Design*, 24–26 Jun 2014, Luxembourg, 289–294.

[9] Atanassova, V., Doukovska, L., Mavrov, D., & Atanassov, K. (2014) InterCriteria Decision Making Approach to EU Member States Competitiveness Analysis: Temporal and Threshold Analysis. P. Angelov et al. (eds.), *Intelligent Systems'2014, Advances in Intelligent Systems and Computing* 322, 95–106.

[10] Atanassova, V., Doukovska, L., Karastoyanov, D., & Capkovic, F. (2014) InterCriteria Decision Making Approach to EU Member States Competitiveness Analysis: Trend Analysis. P. Angelov et al. (eds.), *Intelligent Systems'2014, Advances in Intelligent Systems and Computing* 322, 107–115.