RULE ENGINE-BASED MODULE FOR FINANCIAL ANALYSIS IN DECISION MAKING PROCESS IN PRIVATE UNIVERSITIES

Slavcho Chungurski
FON University Skopje, Republic of Macedonia, slavcho.chungurski@fon.edu.mk
Leonid Djinevski
FON University Skopje, Republic of Macedonia, leonid.djinevski@fon.edu.mk

Abstract: The sustainability of universities, especially for the case when the stockholders is private capital is in constant effort to make worthwhile decisions. As a profit institution a university goal is to base those decisions on deep comprehensive analysis. Manages usually use intuitive reasoning which however useful it can be, it will eventually become overwhelming for one person. Therefore, leaning on accurate, fast calculating quantitative models is the best solution is these cases with large business model workflow. This decision-making process can be widely supported with an appropriate usage of software suitable for creating different scenarios which we have developed in our past work. This paper presents an extension of the computational module that analyzes the viability of each faculty in one university and based on that helps in taking the decision to keep or eliminate a particular department or faculty. The primary goal of the paper is to present the dedicated software application that supports the process to overcome the issue with the courses overlapping between the departments.
Keywords: expression trees; university management; decision making, private universities

INTRODUCTION

Private universities make strategic decisions on daily bases, by evaluating different circumstances that influence the final results. Making reasonable strategic decisions like adding or removing an item from the portfolio of faculties is a choice between the academic quality on one side and the economic viability on the other. Therefore, all possible scenarios have to be reviewed in order to justify the efficiency of a particular scenario, for what we have developed a computational module [1]. This computational module supports universities consisted of several faculties and departments. It covers the academic requirements in number of professors and teaching assistants for the whole university, according to their areas of expertise. The number of the academic staff is then translated in terms of cost for salaries that is compared with the existing revenues from each faculty. Having under consideration that part of the decision-making process is timeliness, the computational module uses flexible spreadsheet models to support their day to day decisions. Thus, automation of the processes data is limited.

Many papers are published in the area of university management, however most of them discuss the qualitative models for decision making in public universities like collegial decision-making model, bureaucratic decision-making model and political decision-making model [2]. Additionally, De Silva and Armstrong **Error! eference source not found.** give an assessment of good governance practices in university context in Australian higher education sector, emphasizing that an effective governance structures play a very important role in attracting most needed funds, and building stakeholder confidence and reputation. Although in relatively old paper [4] discusses some quantitative models for university budget requests and [5] analyze the bureaucratic and coalitional model of budgeting. In [6] Guest, presents an economic framework for analyzing the undergraduate fee and enrolment decisions facing Australian universities in 2005. All these existing papers present backward analysis and draw conclusions based on historical events. According to our knowledge, the paper in [1] is the first to address the topic of computationally producing analysis by creating different scenarios.

However, in this paper, we present extension of the computational module using expression trees, in order to more precisely evaluate all possible details for the academic and financial part of the analysis for supporting the future decision making.

This paper is organized in 4 sections. In Section 2 we present an overview of the existing computational module including description of the input data in the subsections, followed by the description of the extension of the computational module in Section 3. We conclude in Section 4.

THE COMPUTATIONAL MODULE

The main goal of the computational module is to visualize the profits between the academic staff costs and the tuition revenues for each faculty of the university. The size of these profits will show the extent to which the university could feel comfortable with the particular faculty, due to its positive profit, or the extent to which it should put more effort on closing the negative profits by enrolling more students, meaning overcoming the loss and pass the break-even point.

The requirement for the professors and teaching assistants is based on the number of active faculties and number of maximum legally allowed teaching hours per lecturer. The management of teaching hours and active departments, when using the extension of the computation module presented in this paper, can provide the decision makers will immediate insight in the organization of the academic staff.
By including all the configurations in the computational module, different scenarios could be obtained about viability of faculties existence. The data that is included is consisted of all of the University's nationally accredited programs. Departments can be activated by language, year and unit.
By activating a department, the computational module can generate new scenarios over all the configured data for the facultie's programs.
At first, it looks trivial that by simple Subtotal option in the base table can calculate the number of total lecturing hours per vocational areas. Nevertheless, existence of interrelated departments that share same courses, interdisciplinary departments that have courses in their programs from other departments and elective courses that can be chosen from the courses offered in all other active departments, are making thing complicated. Not activating some of the faculties, can introduce a small impact on lowering the number of lecturing hours and through that on lecturers. If a faculty doesn't exist, but its courses are part of the other faculty's programs that are active, the incremental staff number might be insignificant or even zero.
This departmental interdependence leads to the main problem for the module which is course's overlapping. For example, the course Principle of economics is taught at first year of studies in all five departments of the Faculty of economics and all other five faculties in the model. But, teaching this subject requires only 4 teaching hours per week, nevertheless the number of activated faculties and departments. If previously mentioned Subtotal option is applied in this situation and hypothetically nine departments are active where this course is taught, the result will be 4x9 = 36 lecturing hours, which is extremely wrong. Therefore, the challenge here is to find a formula that will remove the overlapping and not multi-counting the course's appearance.
In our module we propose a solution for the problem in the following way. First, in additional column to the base table, using control flow functions the number of the course active appearances that belong to the same unit and language (if the university has different places and languages for performing its activity) is counted.
Second, for each row of the base table, meaning for each course and its each appearance, number of weekly teaching hours per course is divided by the total number of its active appearances in the base (calculated in the first step).
This is the main column of the module, and its use solves the overlapping problem. The way of doing that and its numerical explanation is shown further below.
Before the final report on which the decision will be based, the intermediate one sums the numbers in the column from the previous bullet.
Using this column, nevertheless how many times the course appears as active in the base table, the result calculates it only once. If we further build the explanation on the Principle of economics example from earlier, the model will calculate the following:

- Hypothetically, the course appears active in nine different departments for the main unit and for the national language. In that case, the cell next to each of the course appearance will result in 9. Since, the number of weekly teaching hours for this course is 4 (or 2 + 2, lecturing and exercises), the cells of the main column will calculate 0.4444 (or 0.2222 + 0.2222, lecturing and exercises) each time for the nine appearances. When summing 0.4444 nine times for this course in the report, the result will be 4 teaching hours, regardless the number of faculties from where the students attending it come.
- If the university lowered the number of active departments, again hypothetically, say on five for the main unit and for the national language, the cell next to each of the course appearance will result in 5. Since, the number of weekly teaching hours for this course is always 4 (2 + 2), the cells of the main column will calculate 0.80 (0.40 + 0.40) each time for the five appearances. When summing 0.80 five times for this course in the report, the result again will be 4 teaching hours, regardless the change in the number of active departments. The course is taught only once per week, assuming that the faculty has enough capacity to absorb the new-coming students from the additionally included departments.

Also, teaching hours are divided per winter (1st, 3rd and 5th) and summer (2nd, 4th and 6th) semester.
In order to get the number of academic staff, instead of number of teaching hours, the last obtained figure should be divided by the maximum number per lecturer variable for both winter and summer semesters. Since, the maximum number per lecturer is negotiable variable; by changing it, the number of lecturers will be changed.
The separation of required teaching hours per semester was done in order to get the number of academic staff needed per semester. If for instance, the number of teaching hours per semester in the vocational area of economy is 141 and 159 and the maximum number of teaching hours is 8 for professors and 16 for teaching assistants, the university will need 9 professors and 5 teaching assistants in the winter semester and 10 professors and 5 teaching assistants in the summer semester. But, there is a mistake in this reasoning. Since, the

courses are taught only during one semester, the academic staff is free to take new courses in the next semester after ending on one of them. Therefore, the average number between semesters is taken. Not the higher one, since if there is big gap between semesters in academic staff, the faculty will consider smoothing the particular program.

Cost for the required number of the academic staff

Number of the academic staff multiplied by the average salary variable will give the amount of the university cost for academic salaries. By changing the average salary, when using the model presented in this paper, the decision makers will immediately have an insight in the new amount of this cost. Departmental interrelations mentioned earlier have also an effect on the faculty costs.

Revenues from the tuitions

The number of students per faculty multiplied by the average tuition for that faculty (having into considerations the scholarships, partial and in total) gives the amount of revenues per faculty. With adding a new faculty, every student enrolled on that faculty is additional revenue for the university. Aside of that, not every teaching hour for that program is additional cost, since some of the courses are already taught in some of the existing faculties.

The use of the module

This model was created for real needs of a private university in order to give an insight in the justification of adding or shutting some of the faculties or departments. Its first purpose was to show the necessary number of the academic staff, considering that higher number of lecturers means higher costs for the university.

Then, the management board has decided that the change in needs for academic staff doesn't mean anything without taking into consideration the number of students enrolled in that particular faculty and their average tuition. Therefore, the model was upgraded with the financial part.

The model doesn't deal with the profitability of each faculty. It serves only to the decision making purposes and this process draws its conclusions from the analysis of incremental amounts. Profitability means higher revenues than costs for each faculty. In this concept, if faculties share same courses, they divide the costs for these courses between them. Following this calculation, some of the faculties might show endangered profitability, or even negative one. The disadvantage of this concept in decision making is its partiality. It means that it doesn't observe the university in a holistic way i.e. if one faculty is removed due to its negative results; the university cuts the revenues from this faculty, but not its total costs. In the way these costs are calculated, it doesn't mean they are cut by cutting one faculty. Some of the courses might further exist in other departments, which profitability will be decreased.

If we go back to the example with the course Principle of economics when it was shared by nine departments, then each of them will share a ninth of the cost for lecturing this course. If one of the departments is being removed, in the profitability analysis appears that some cost is eliminated. But, in fact it is not true. Eight departments where this course is taught still exist and now they share one eight of the unchanged cost. They will show lower profitability after removing the mentioned department.

For those reasons, university and all other institutions should be reasoning in incremental amounts, i.e. bearing on mind only the additional effects in revenues and costs that the university has. One more hypothetical example will illustrate the situation. Let's say that there is a faculty that has negative results according to profitability analysis (revenues 100.000 euro and costs 120.000 euro). The costs' amount is calculated in a way it was shown in previous paragraph. If the decision is made based on this analysis, the management will shut down this faculty. But, there might be situations where the program is interdisciplinary and consisted of courses that are already taught on other programs. By shutting down the program, the cost will not be removed, but the revenues will. Thus, in the decision making process, the existence of this faculty will show positive change in revenues, but zero change in costs. Therefore, it will stay as a part of the university offer.

After creating this model, when comparing the costs and the revenues explained in the previous section, was noticed that sometimes incremental costs are that low that is pay off to add the new department or if it exists than worth keeping it even with the low number of students.

RULE ENGINE

Required flexibility is solved by implementation of special parser that can parse any kind of expressions: Boolean, numeric, comparisons, text operations etc. This parser is implemented in special C# library RuleEngine, where one enumerator and two public classes are defined:

- Static Class RuleEngine that preprocesses the input string and executes the calculation. Its public method Calculate accepts the input string with the expression and returns a pair of string that contains the result value and an expression type that is evaluated from the expression.
- Enum ExpressionType with possible values: BOOLEAN, NUMERIC, LITERAL
- Class ExpressionTree which contains the whole logic for building expressions trees and their calculation, upon the input string

By definition, this parser accepts every character but only the characters represented in (and combinations) are accepted as an operators or special symbols.

Table 1. Supported operations in Rule Engine

Order	Operator	Operation	Result	Operators
1	>=	greater than or equal	Boolean	Numeric or string values
2	<=	less than or equal	Boolean	Numeric or string values
3	==	equal	Boolean	Numeric or string values
4	!=	not equal	Boolean	Numeric or string values
5	>	greater than	Boolean	Numeric or string values
6	<	less than	Boolean	Numeric or string values
7	|	Logic OR	Boolean	Boolean values
8	&	Logic AND	Boolean	Boolean values
9	!	Logic NOT	Boolean	Boolean values
10	+	Sum	Numeric	Numeric or string values
11	-	Subtract	Numeric	Numeric values
12	*	Multiply	Numeric	Numeric values
13	/	Division	Numeric	Numeric values
14	contains	left contains right	Boolean	String values
15	starts with	left starts with right	Boolean	String values
16	ends with	left ends with right	Boolean	String values
17	"	string qualifier	/	
18	\	escape character	/	

There are three types of expressions by their appliance. The first one is the type that is used in sales part, the second one is used in engineering part and the third one is used in the production part. Therefore, there are three separate collections of expressions. Every expression can reference another expression in the process of its evaluation, as well as there can be a reference to some literal that can be defined as reference to a property, or reference to some value.

RuleEngine class
This class is static class that has the following properties and methods:

```
public static Dictionary<string, char> Operations;
```

This dictionary is used to replace multi-character operators with single character, for simplifying the algorithm. This means that the single - characters are not allowed in the literals, but proper characters from the ASCII table that are used for ASCII graphics are used, to avoid issues.

```
private static string ReplaceKeysWithValues(string ex)
```

This method applies the dictionary on the input string ex, which means it replaces the multi-character operations with single characters

    public static KeyValuePair<string, ExpressionType> Calculate(string expression)

This public method represents an interface to the recursive algorithm for Expression Tree. It returns a result and the Expression Type, both in a key-value pair. This method also is doing preprocessing of the strings used in the expressions. This means that the strings inside the expression are recognized and replaced with appropriate literals, before they are send as string collection together with the expression in the ExpressionTree class.

ExpressionTree class

The main logic for calculation of expressions is implemented in this class. It represents a recursive node from binary tree. This class has the following properties:

    public string Value
    public ExpressionTree LeftSide
    public ExpressionTree RightSide
    public ExpressionType ResultType

As it is represented, each node has its Value stored as string and which is parsed lately, if needed, and there are left and right side which represents that the node has two child nodes. All of its operations are considered as binary, which means that all nodes have exactly 2 children, except literals (operands) that are without children. The only supported non-binary operation is Logical NOT, and it is resolved as a node where the left child is NULL and the right one is the expression that is negated. Finally, expression type is automatically determined, depending on the operation stored in the Value, as well as the expression types in the child nodes.

There is one additional public property

    public static List<string> stringList

which is static list that contain strings that will be used to evaluate the strings from the expression. Actually, this string list is populated from the outside, in the preprocessing phase of evaluation, from RuleEngine class and it contains the strings that are identified in the preprocessing phase.

The class contains the following methods

    public ExpressionTree(string ex)

This method represents the class constructor that constructs the node recursively based on the input expression. It is important to mention for this constructor that if the expression starts with – (minus) sign, it is enclosed with parentheses with addition of 0 before it, for example: -5 will be replaced with (0-5). This is a must, to resolve the negative values, considering that the sign – (minus) is used both as operator as well as sign for negation. With this solution, all negative values are replaced with subtract operation from 0.

    private List<string> SplitExpression(string ex)

One of the main methods in this module. A method that finds the first main operation and divide the expression on three parts:

- left side of the operation,
- operation itself and
- right side of the operation that can be composed of more main operations

The right side is then sent to recursion, which means that arithmetic and logical prioritization of the operations is not working. This means that 5+2*2 will be calculated as 14, and not 9. Because of this, proper grouping is essential for this algorithm, but considering that the expressions will be developed by a tool, this grouping can be easily implemented.

    public string Calculate()

This method recursively calculates the value of the expression. Calculation of the left side and calculation of the right side are combined with the appropriate operation, stored in the node. Terminal cases are when the node is NULL, or it is atomic – leaf in the tree.

    private string Evaluate(string v)

This method evaluates a value from the input string. This method will pull the values from the value collections, like database values, parameters, identified strings, another expressions etc., depending on the format of the input value. The input value cannot contain any of the operators, but it is evaluated before this method is called.

    public bool isAtomic()

Inspects if the current node is atomic - without children (leaf), and returns an appropriate bool value.

    private bool isLiteral(string l)

Inspects if the input string is literal, which means that it doesn't contain any operator

    private string RemoveSurroundingParentheses(string ex)

If the input string expression is enclosed in parentheses, they are removed with this method

```
internal void AddStrings(List<string> quoted)
```
A method that is called from pre-processing phase to add the identified strings in the expression in the stringList collection.
There are also a few extensions of the String class, that are used in the whole namespace.
```
public static bool ContainsAnyValue(this String str, IEnumerable<string> vals)
```

Checks if the string contains any string value from the collection vals.
```
public static bool ContainsAnyValue(this String str, IEnumerable<char> vals)
```
Checks if the string contains any char value from the collection vals.
```
public static string[] Split(this String str, string divider)
```
Splits the string using string divider. Same as Split which works with char, but this is with string.
```
public static double AsDouble(this String str)
```
Returns double value parsed from the string.
```
public static bool AsBool(this String str)
```
Returns bool value parsed from the string.
```
public static bool IsNumeric(this string s)
```
Checks if the string is numeric, and returns corresponding bool value.

CONCLUSION AND FUTURE WORK

The goal of this paper was to present the comprehensive response of the results in the extension of the computational module by implementing the rule engine which is based on expression trees data structures. By using this extension, university managers could become able to making even more founded decisions, which can contribute to taking reasonable actions without time delays.
This paper had presented only prove of concept for one scenario for determined number of faculties, maximum teaching hours per lecturer, assumed amount of salary and current number of students and tuition. However, the rule engine takes in consideration the parameters change (for example if the university lower the salary), and generates results that will lead to appropriate conclusions for decision making. The extension, for compared to the computational module, offers results that are 100% customized by the university stakeholders, as long as it is possible to configure the input data.
As future work we plan to continue our research in implementing this extension of the computational module in real world scenarios.

REFERENCES

[1]. M. Ivanovska, L. Djinevski, S. Arsenovski, “Computational Module as a Tool for Financial Viability Analysis in Decision Making Process in Private University Institutions”, ICT Innovations 2014, Ohrid, Macedonia, 12th – 14th Septeber 2014.
[2]. Khefacha, I., B.L.: Decision-making models and university governance: Experience from tunisian public higher education establishments. In: Proceedings of the European Conference on Management, Leadership and Governance. pp. 53{60 (2009)
[3]. De Silva, C., Armstrong, A.: Evaluation of corporate governance measures: An application to the australian higher education sector. Journal of Business Systems, Governance & Ethics 7(1) (2012)
[4]. Heinze, D.C.: Decision models for university budget requests. Journal of Financial and Quantitative Analysis 6(03), 1035{1040 (1971)
[5]. Hills, F.S., Mahoney, T.A.: University budgets and organizational decision making. Administrative Science Quarterly pp. 454-465 (1978)
[6]. Guest, R.: The undergraduate fee and enrolment decisions facing Australian universities from 2005. In: Education Economics. pp. 59-73 (2006)