A heuristic approach for resolving the Class Responsibility Assignment Case

Maximiliano Vela
maximiliano.vela@hotmail.com

Yngve Lamo
yla@hib.no

Fazle Rabbi
fra@hib.no

Fernando Macías
fmac@hib.no

Bergen University College
Bergen, Norway

Abstract

This paper describes one of the solutions for the ninth Transformation Tool Contest (TTC '16)[1], which resolves the Class Responsibility Assignment Case using a transformation tool based on Microsoft Excel and Visual Basic. In this project, these relatively unusual technologies are used to effectively enhance the processing of large models and matrices throughout different test cases proposed for the competition. Given a Responsibility Dependency Graph, the solution analyzes the relationships among the given features and produces a class diagram that aims to get the highest possible cohesion and lowest possible coupling. The solution is based on an adaptation of the Markov Clustering Algorithm used to manage bi-directional, un-weighted sets of nodes and grouping them into clusters. The aim of this work is to take one step further in the graph/model optimization field through the treatment of matrices, a strategy that is not so widely used.

1 Introduction

Cohesion is a property that refers to the degree to which the elements of a module belong together, while coupling is the degree of interdependence between different modules. In order to generate high-quality models, the need for measuring and comparing how strong the relationship is between different components of a module is extremely important. Even though in some cases the coupling will always be inevitably higher than the cohesion, there will always exist ways to optimize the resulting model as much as possible.

The algorithm used in this solution to generate the resulting class diagram is based on the foundations of the Markov Clustering Algorithm [SVD00] [KM09]. However, most of the steps have been slightly or completely altered since they have been deemed not as effective for the proposed problem.

As described in the problem definition, for a total of 18 input features there are 682.076.806.159 possible class diagrams built upon different feature combinations, so the time it takes for the solution to generate a potentially optimal output is essential.

The objective of the proposed solution[2] is to perform a model transformation from the initial Responsibility

In: A. Garcia-Dominguez, F. Krikava and L. Rose (eds.): Proceedings of the 9th Transformation Tool Contest, Vienna, Austria, 08-07-2016, published at http://ceur-ws.org

[1] The full problem description can be found at: github.com/martin-fleck/cra-ttc2016/

[2] The proposed solution and other useful files can be found at: github.com/maximiliano-vela/ttc-2016 - SHARE demonstration/reviewing VM: XP-TUe_XP_Excel_v2.vdi @ http://is.ieis.tue.nl/staff/pvgorp/share/

Dependency Graph received as input into a Class Diagram which attempts to achieve the highest possible CRA-Index by having as high cohesion and as low coupling as possible.

The meta-model transformation relies in linking each feature (method or attribute) to a class that encapsulates it, as shown in Figure 1.

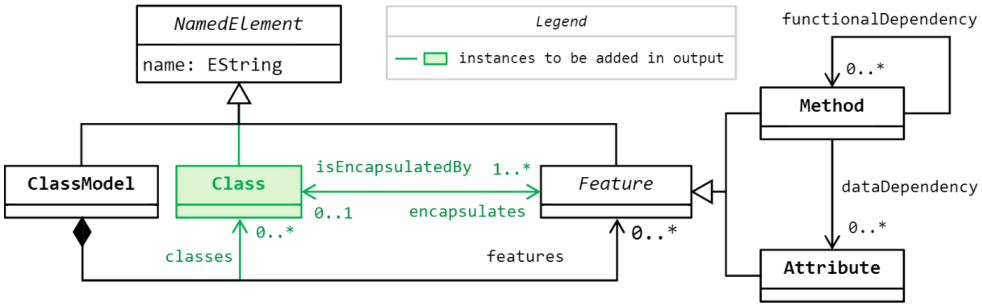

Figure 1: Responsibility Dependency Graph meta-model[3]

2 Motivation and Features

At the beginning of this project, the idea of using Microsoft Excel as the main part of the transformation was merely a result of the developer's working experience with Visual Basic in different fields of the industry. However, since the moment we found out about the Markov Clustering Algorithm and its use within matrix/graph operations, Excel naturally started to sound more and more convenient. Additionally, since matrices are worked upon the actual spreadsheet, the debugging of the algorithm as well as further experimentation with it became significantly easier. The tool is highly transparent, very easy to use, requires little to no technology stack at all (for Windows users, that is) and its transformation process, described in the following section, is relatively simple.

3 Algorithm Implementation

The implementation starts off with a pre-processing step that transforms the input file into meaningful data that will shortly after be used in order to generate the classes. The results are finally formatted as an .xmi file for further use. Several inputs have been tested for the competition, however Input A (the smallest one) will be the one used as an example in this section.

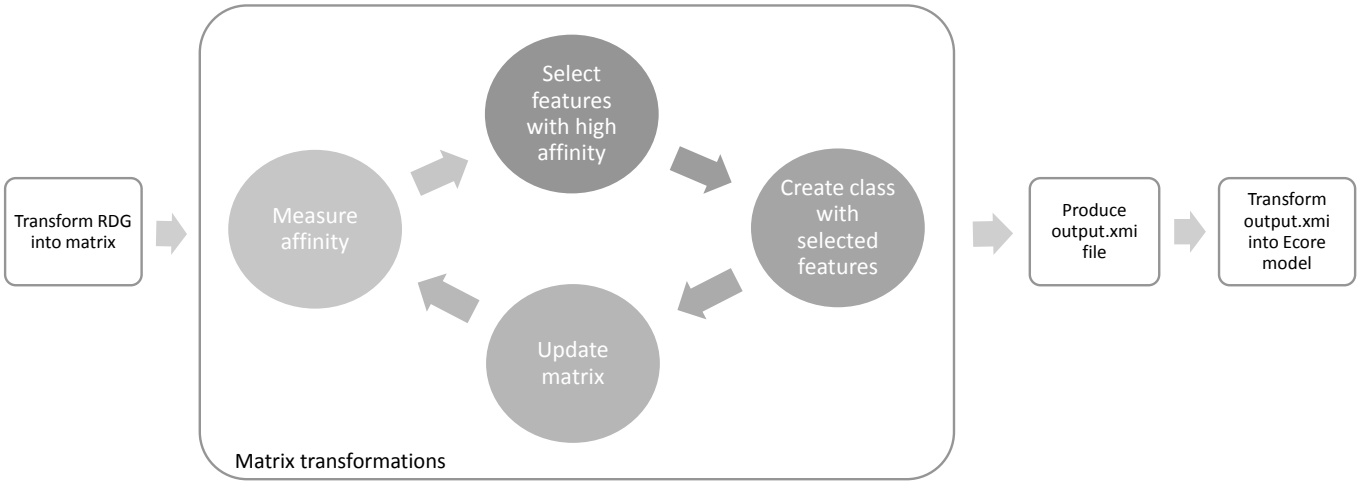

Figure 2: Transformation process

[3]Taken directly from the full case description, which can be found at https://github.com/martin-fleck/cra-ttc2016/blob/master/case_description/TTC2016_CRA.pdf

3.1 Initial Processing

The first step in the process is importing the Responsibility Diagram and parsing it in order to harvest the available features and existing relationships between them.

A matrix of size $\rho \times (\rho + \sigma)$ is generated, where ρ is the number of methods and σ is the number of attributes. The matrix will be filled with 0, 1 or 2 which represents the type of relationship between two particular features (no relationship, uni-directional and bi-directional respectively). This value will be referred to as α. The result after this step for Input A is shown in Table 1. In the example, relationships for each method with itself will be 1 but this can vary depending on certain properties (this concept is explained further under section 4).

Table 1 Associate Matrix for Input A - First Step

	Methods				Attributes				
	0	1	2	3	4	5	6	7	8
0	1	0	1	0	0	0	0	1	1
1	1	1	0	2	0	0	1	1	0
2	0	0	1	1	1	0	0	0	0
3	1	2	0	1	0	1	0	1	0

The distinction between uni- and bi-directional relationships in the associate matrix is the first difference from the classic Markov Clustering Algorithm implementation, which contemplates bi-directional links only.

Table 2 Associate Matrix for Input A - Second Step

	Methods				Attributes				
	0	1	2	3	4	5	6	7	8
0	1,4	0,2	1,1	0,2	0	0	0	1	1
1	1,2	1,5	0,1	2,4	0	0	1	1	0
2	0,1	0,1	1,3	1,1	1	0	0	0	0
3	1,2	2,4	0,1	1,5	0	1	0	1	0

In the next step, the algorithm analyzes the similarities between each pair of methods by counting how many features they share relationships with (both values should be > 0 to be counted). In the example, methods 1 and 3 have four features in common (0, 1, 3 and 7). Each value will be calculated for every pair of methods and placed next to each cell value separated by a comma, and will be referred to as β. This newly calculated value is arguably one of the most important across the entire algorithm, as it directly related to the stregth of the relationship between two given methods. As a result, values in each cell will correspond to α,β as shown in Table 2. Attribute values will remain unchanged in this step.

Shortly after, a new value δ will be calculated on each of the recently modified cells. This calculation, as well as the use of the diagonal will vary depending on various properties associated with the initial Responsibility Dependency Graph. This is explained more in depth in section 4.

However, in this case the affinity index (δ) will be defined as $\delta = \beta$. Results for this example can be found in table 3.

3.2 Class Generation

In order to achieve the last step of the algorithm and finally get to the resulting class diagram, the following calculations will be performed:

1. Sum total for each method row, showing which method has the most/strongest relationships.
2. Maximum affinity index -calculated previously- for each method column, showing which pair has the strongest relationship.
3. Sum total for each attribute column, showing by how many methods each attribute is used.

Results after this step are shown in Table 3.

From here on, we start by choosing the first row with the highest row total. If the matrix exceeds a specified threshold in the number of methods, values in this row will be slightly increased in order to simulate the generation of a superclass or "leading" class, but in the example shown values will remain the same.

Table 3 Associate Matrix for Input A - Third Step

	Methods				Attributes					Sum
	0	1	2	3	4	5	6	7	8	
0	4	2	1	2	0	0	0	1	1	11
1	2	5	1	4	0	0	1	1	0	14
2	1	1	3	1	1	0	0	0	0	7
3	2	4	1	5	0	1	0	1	0	14
Max/Sum	4	5	3	5	1	1	1	3	1	

For each method column in the row selected before, we check if the affinity index is equal to the previously calculated maximum. If the values are

the same, we store that method in an array in order to group them together in the class generation step. In the example, the first row selected will be Method 1 (with 14 as row sum), and the only candidate method will be itself. This attempts to ensure the highest possible cohesion within all method-to-method relationships.

After that we move on to the grouping of attributes to be put within the same class. In order to do so, we calculate the sum total of each attribute column but only for the methods previously selected. If the sum is equal to or greater than 50 percent of the column total, that attribute will be selected since the majority of methods that use it are candidates for the newly generated class. If the sum is less than 50 percent that means the attribute will produce higher cohesion placed in a different class. For Method 1, attribute 6 will be grouped with it since $1 \geq 0.5 \times 1$, but attribute 7 will not since $1 < 0.5 \times 3$.

Methods and attributes selected in the last section will be placed in a newly created class. In order to prevent other classes from re-using them in the future, column values for each attribute and row/column values for each method will be set to zero. This process will go on generating classes by grouping methods and attributes until values on all cells are zero.

After wrapping up the last set of features within a class, the solution will transform the class diagram into a .xmi file with the correct formatting and store it within the same file-path as the input.

4 The Diagonal Dilemma, Affinity Index Calculation and Leading Class

When analyzing and processing affinity indexes, values shown in the diagonal are often (such as in the example shown in this paper) necessary in order to generate a class diagram that carries a high CRA-index. However, in some cases it has been proven to be misleading and therefore largely increasing the overall coupling of the output.

For this reason, it has been determined the need to establish a threshold within the number of methods required in order to deem the diagonal counter-productive. For our solution, every time the input exceeds 12 methods the values in the diagonal will become zero. This threshold is not completely precise and might be refined after further experimentation.

Furthermore, for inputs in which the number of methods is < 12 the diagonal might require to be turned to zero as well for very specific cases (such as having less attributes than methods and thus requiring multiple methods to be put together in order to increase the cohesion). This avoids methods to be isolated and being grouped up only with the attributes they make use of.

When it comes to calculating the Affinity Index δ, several different heuristics using both α and β have been tested (such as $\alpha + \beta, \alpha^2 + \beta$, etc.) but the best results have been obtained by using simply $\delta = \beta$.

Another concept introduced in this algorithm is the generation of a main or leading class which contains the majority of the methods/attributes, while keeping the overall strongest relationships between features in the associate matrix to classes other than the main one. This is a means to represent the way in which developers generally build class diagrams when illustrating real-life schemes, having one class that contains a large number of methods and attributes (i.e. class Person/Student/Employee, class Product/Furniture/Vehicle, etc.) as well as a few other classes that provide functionality, usability or additional relevant information.

5 CRA Index and Execution Time

The results obtained by the proposed solution can be found below:

	Proposed Excel Solution[4]		TTC MOMoT Solution	
Input	CRA-Index	Execution Time	CRA-Index	Execution Time
A	3.0	0,46 sec.	3.0	4 min. 03 sec.
B	2.99999999	1,13 sec.	1.125	5 min. 05 sec.
C	0.64803921	1,92 sec.	-5.63571428	12 min. 02 sec.
D	-0.2913547	7,54 sec.	-23.6338095	26 min. 51 sec.
E	0.21916410	21,83 sec.	-69.65545274	38 min. 08 sec.
F	1.92302499	2 min. 46 sec.	Unknown	Unknown
G	-0.0343154	37 min. 42 sec.	Unknown	Unknown

Table 5 Class Diagram Generation Results for Proposed Solution compared to TTC's

[4]Executions have been performed on a Lenovo Y50 notebook running Windows 8 x64, i5 4210H @ 2.90GHz, 8.00GB

6 Visualization

The output .xmi file is finally transformed into an ecore model which can be later used for visualization through "Initialize ecore_diagram diagram file" feature in Eclipse. Figure 3 shows the class diagram for the output produced for Input A.

The proposed solution can be executed in a remote server through a Powershell prompt, allowing users to provide an input file and transform it into an output .xmi file or an ecore model.

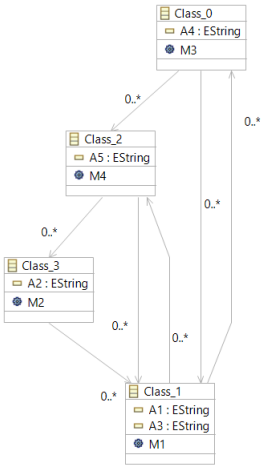

Figure 3: Class Diagram for Input A

7 Related work

There have been several attempts done by researchers to solve the CRA problem. Recently, meta-heuristic optimization techniques such as Genetic Algorithm (GA), Hill Climbing (HC), Simulated Annealing (SA), Particle Swarm Optimization (PSO) have been analyzed by many research groups [E07]. Multi-objective genetic algorithm (MOGA) is another technique that has been used to solve the Class Responsibility Assignment Case [BBL10] [MJ14].

8 Conclusion

Although the Markov Clustering Algorithm has been proven to be exceptionally useful for finding strongly connected components within bi-directional, un-weighted graphs, a classic implementation of this algorithm is not enough to completely cover the problem analyzed in this paper. Originally this algorithm merely compares similarities between features (which belong to an unique type), and expands the variables until they're stable enough to point out the clusters found in the graph. In-depth explanation of this algorithm can be found in its very own author's paper [SVD00].

Without the adjustments proposed in this paper, but specifically those described under section 4, Excel would simply return poorly generated class diagrams and the resulting CRA-index would be considerably lower.

When it comes to resulting CRA values, they're closely comparable to those obtained by other solutions in the competition, being located under highly accurate transformation tools such as VIATRA or Henshin. However these solutions are based on space-exploration strategies and genetic algorithms respectively, and therefore their execution time is significantly higher as they require to wander through several solutions and compare them afterwards.

Even though the results obtained are somewhat respectable, there still exists a gap between the class diagrams generated and the overall optimal class diagram for each input, which remains yet undisclosed. This gap could be filled by refining the proposed techniques, establishing new thresholds or simply adding new variables that could help achieve higher CRA-indexes.

References

[SVD00] S. M. van Dongen. Graph Clustering by Flow Simulation. 2000.

[KM09] K. Macropol. Clustering on Graphs: The Markov Clustering Algorithm. 2009.

[E07] A. P. Engelbrecht. Computational Intelligence: An Introduction, 2nd ed. John Willey & Sons. 2007

[BBL10] M. Bowman, L.C. Briand, Y. Labiche. Solving the Class Responsibility Assignment Problem in Object-Oriented Analysis with Multi-Objective Genetic Algorithms in IEEE Transactions on Software Engineering, vol. 36, no. 6, pp. 817-837, 2010

[MJ14] H. Masoud, S. Jalili. A clustering-based model for class responsibility assignment problem in object-oriented analysis in Journal of Systems and Software, volume 93, pp. 110-131, 2014