Identification of Potential Classes in Procedural Code Using a Genetic Algorithm

Farshad Ghassemi Toosi
Lero, University of Limerick
Limerick, Ireland
farshad.toosi@lero.ie

Asanka Wasala
Lero, University of Limerick
Limerick, Ireland
Asanka.Wasala@lero.ie

Goetz Botterweck
Lero, University of Limerick
Limerick, Ireland
goetz.botterweck@lero.ie

Jim Buckley
CSIS, University of Limerick
Limerick, Ireland
Jim.Buckley@ul.ie

ABSTRACT

We present a novel approach for discovering and suggesting classes/objects in legacy/procedural code, based on a genetic algorithm. Initially, a (procedures-accessing-variables) matrix is extracted from the code and converted into a square matrix. This matrix highlights the variable-relationships between procedures and is used as input to a genetic algorithm. The output of the genetic algorithm is then visually encoded using a heat-map. The developers can then (1) either manually identify objects in the presented heat-map or (2) use an automated detection algorithm that suggests objects. We compare our results with previous work.

CCS CONCEPTS

• **Software and its engineering → Object oriented architectures**;
• **Mathematics of computing →** *Permutations and combinations*;

ACM Reference Format:
Farshad Ghassemi Toosi, Asanka Wasala, Goetz Botterweck, and Jim Buckley. 2018. Identification of Potential Classes in Procedural Code Using a Genetic Algorithm. In *GECCO '18 Companion: Genetic and Evolutionary Computation Conference Companion, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3205651.3205720

1 RELATED WORK AND BACKGROUND

Literature [2, 9] suggests that maintenance of Object Oriented (OO) software is easier than that of procedural software. This is an important advantage, because the effort consumed in software maintenance and evolution can dwarf the original effort consumed during initial development [7]. Although most currently developed software systems are OO, there is still a large amount of procedural code in valuable, mature, legacy systems that companies wish to migrate to the OO paradigm. This is the problem faced by our commercial partner, and the work addressed in this research.

The term 'objectification' has been applied to this migration and can be approached by identifying a set of procedures that access a common state. Literature suggests that objectification of code has several advantages including easing maintenance and testing [8].

A number of authors have leveraged concept analysis [1, 4] towards this goal [3, 5, 8] semi-automatically. For example, Siff and Reps [8] propose a semi-automated framework where they identify the maximum collection of functions sharing common data.

They highlight the utility of the proposed approach in classifying functions into classes. However, they also mention that, depending on the complexity of the programs, the concept lattice can get very large and result in a very large number of concept partitions, thus making manual analysis complex and tedious, a finding consistent with the other research cited and one addressed in this research.

2 METHODOLOGY

Let P be a set of procedures $P = \{p_1, p_2, ..., p_8\}$ and V be a set of all global variables $V = \{v_1, v_2, ..., v_7\}$ in a given system. The matrix PV in Table 1 on the left shows their inter-dependencies.

Table 1: Procedure-Variable (PV) Dependency Matrix on the left, Procedure-Procedure (PP) Matrix on the right.

	V_1	V_2	V_3	V_4	V_5	V_6
P_1	0	0	1	0	0	1
P_2	0	1	0	1	0	1
P_3	1	0	0	0	1	0
P_4	0	1	0	1	0	1
P_5	1	0	0	0	1	0
P_6	0	0	1	0	0	1
P_7	0	1	0	1	0	1
P_8	1	0	0	0	1	0

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
P_1	2	1	0	1	0	2	1	0
P_2	1	3	0	3	0	1	3	0
P_3	0	0	2	0	2	0	0	2
P_4	1	3	0	3	0	1	3	0
P_5	0	0	2	0	2	0	0	2
P_6	2	1	0	1	0	2	1	0
P_7	1	3	0	3	0	1	3	0
P_8	0	0	2	0	2	0	0	2

Table 2: Procedure-Procedure (PP) Matrix corresponds to: $\{P_5, P_8, P_3, P_2, P_7, P_4, P_1, P_6\}$.

	P_5	P_8	P_3	P_2	P_7	P_4	P_1	P_6
P_5	2	2	2	0	0	0	0	0
P_8	2	2	2	0	0	0	0	0
P_3	2	2	2	0	0	0	0	0
P_2	0	0	0	3	3	3	1	1
P_7	0	0	0	3	3	3	1	1
P_4	0	0	0	3	3	3	1	1
P_1	0	0	0	1	1	1	2	2
P_6	0	0	0	1	1	1	2	2

2.1 Matrix Preparation

The input matrix for our GA is a square matrix ($|P| \times |P|$). Therefore the PV matrix is multiplied by its transpose ($PV \times PV^T$) and results in a new matrix that we call it PP, (see Table 1 on the right).

GECCO '18 Companion, July 15–19, 2018, Kyoto, Japan

ACM ISBN 978-1-4503-5764-7/18/07.
https://doi.org/10.1145/3205651.3205720

In some cases that are procedures that do not use any variables. Consequently, they are not associated with any other procedure, based on their used-variables. This situation usually happens when the number of variables is much smaller than the number of procedures. Siff *et al.* [8] suggest using additional variables ("does-not-use-variable") in the original matrix to make associations between such procedures.

2.2 Genetic Algorithm Implementation

A set of procedures, P, has a base arrangement $A_P = \{P_1, P_2, \ldots, P_{|P|}\}$ and a base square matrix A_{PP} corresponds to A_P (see Table 1, left and right). The set of P can have $|P|!$ different linear arrangements and, for each new arrangement, the base square matrix is re-arranged accordingly (see Table 2). This problem is known as Optimal Linear Arrangement (also referred to as MinLA or MLA). We propose a GA to solve the problem of MinLA for a set of procedures P and a procedure-procedure matrix PP. The aim of this GA is to produce an arrangement of procedures (A_P) and its corresponding matrix (A_{PP}) so that those procedures with higher similarity (higher value in PP) are placed closer to each other. The corresponding matrix to such an arrangement is converted into a *Heat-Map* so the borders between groups will be clearly distinguished (see Table 2). An arrangement is said to be a good arrangement if similar procedures are placed next to each other.

The fitness function of our presented GA evaluates the goodness of a procedure arrangement (A_P) using its corresponding matrix (A_{PP}). A matrix with high-value cells around the diagonal line, reveals some groups of similar procedures next to each other (see Table 2). We make use the following fitness function: $\sum_{i=1}^{|P|} \sum_{j=1}^{|P|} A_{PP}[A_P[i]][A_P[j]] \times |i - j|$, in order to evaluate how well a A_P is arranged.

The proposed GA in this work applies an Elitism selection technique along with crossover (Cr) *(partially mapped crossover (PMX))*, mutation (Mu) *(exchange mutation (EM))* and reproduction (Re) with the following probabilities $Cr = 25\%$, $Mu = 25\%$ and $Re = 50\%$ [6]. The presented GA in this work tries to find a pair pf A_P and A_{PP} with the lowest fitness cost. We, experimentally, decided to choose the following values for the parameters in the presented GA: $Population.size = 200$, $No.Of.Generations = |N| \times 100$. Once the genetic algorithm terminates, a pair of A_P and A_{PP} is resulted. A_{PP} contains segments where each segment potentially represents an object. The heat-map visually reveals those segments and assists developers in the task of objectification. Additionally, we introduce an automated technique that suggests a number of groups of procedures with shared variables. A threshold is decided and the neighbours of the diagonal line ($A_{PP}[i][j]$ where $i = j + 1$) of the matrix is traversed. If any cell has a value greater than or equal to the threshold then the corresponding procedures to that cell are grouped together and if a cell with a value less than threshold is met then the previous group is sealed and a new group is created; this process continues until all procedures are traversed. We set the value of the threshold at the median of the list of values in A_{PP}. Table 2 suggests three objects as follows: Obj1 = $\{p_5, p_8, p_3\}$, Obj2 = $\{p_2, p_7, p_4\}$, Obj3 = $\{p_1, p_6\}$.

3 CASE STUDY: MODULA-2

In order to illustrate the utility of our research in this work we apply our technique to a modularization problem on *Modula-2*. This case study has been already tested by Lindig *et al.* [5].

Figure 1 shows the heat-map resultant from our GA. As it is shown, there are some blocks of cells nearby the diagonal, in which the object detection algorithm discovers 9 modules as follows: (11, 13, 12, 10, 18, 17, 15, 16, 9, 14) - (28, 31, 25, 27, 26, 30, 29) -(20, 21, 24, 23, 22)- (3, 4, 5) -(2, 1, 0) - (8, 7) - (6) - (32)- (19).

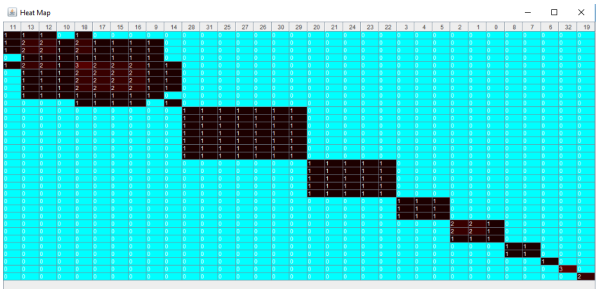

Figure 1: The heat-map of the GA result on *Modula2* source code.

As mentioned earlier, this case study has also been used by Lindig *et al.* [5]. Their proposed modularization suggests the following modules: (0, 1, 2) - (3, 4, 5) - (25, 26, 27, 28, 29, 30, 31) - (32) - (6) - (7, 8) - (19) - (20, 21, 22, 23, 24) - (9, 10, 11, 12, 13, 14, 15, 16 , 17, 18). That is, their suggested results are exactly the same as the results suggested by our technique. One of the advantages of our technique over their technique is the minimum amount of human interaction in the determination of the objects.

Acknowledgement. This work is supported, in part, by Science Foundation Ireland grant 13/RC/2094.

REFERENCES

[1] G. Birkhoff. *Lattice Theory*. Number v. 25, pt. 2 in American Mathematical Society colloquium publications. American Mathematical Society, 1940.
[2] C. L. Corritore and S. Wiedenbeck. Mental representations of expert procedural and object-oriented programmers in a software maintenance task. *International Journal of Human-Computer Studies*, 50(1):61 – 83, 1999.
[3] A. De Lucia G. Canfora, A. Cimitile and G. A. Di Lucca. A case study of applying an eclectic approach to identify objects in code. In *7th International Workshop on Program Comprehension (IWPC '99), May 5-7, 1999 - Pittsburgh, PA, USA*, pages 136–143. IEEE Computer Society, 1999.
[4] B. Ganter and R. Wille. *Formal concept analysis: mathematical foundations*. Springer Science & Business Media, 2012.
[5] C. Lindig and G. Snelting. Assessing modular structure of legacy code based on mathematical concept analysis. In *Proceedings of the 19th International Conference on Software Engineering*, ICSE '97, pages 349–359, New York, NY, USA, 1997. ACM.
[6] R. H. Murga I.Inza P. Larrañaga, C. M. H. Kuijpers and S. Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artif. Intell. Rev.*, 13(2):129–170, April 1999.
[7] V. Suma S. Christa, V. Madhusudhan and R. Jawahar J. Software maintenance: From the perspective of effort and cost requirement. In *Proceedings of the International Conference on Data Engineering and Communication Technology*, pages 759–768. Springer, 2017.
[8] M. Siff and T.W. Reps. Identifying modules via concept analysis. *IEEE Trans. Software Eng.*, 25(6):749–768, 1999.
[9] Susan Wiedenbeck, Vennila Ramalingam, Suseela Sarasamma, and CynthiaL Corritore. A comparison of the comprehension of object-oriented and procedural programs by novice programmers. *Interacting with Computers*, 11(3):255 – 282, 1999.