Genetic Algorithm for Forming Student Groups Based on Heterogeneous Grouping

ANON SUKSTRIENWONG
Information Technology Department
School of Science and Technology
Bangkok University
40/4 Rama IV Rd., Klong-Toey, Bangkok
THAILAND
anon.su@bu.ac.th

Abstract: - Student group formation plays a critical role in terms of enhancing the success of academic learning. It involves structuring groups in order to help students to learn together. Due to the heterogeneity of students in classes, the process of forming student groups is becoming more complex. In this paper, we proposed an approach to form student groups at a university for software development project to ensure that software products can be delivered successfully on time. Our proposed algorithm is called Genetic Algorithm for Heterogeneous Grouping (GAHG). The algorithm aims to achieve both fairness in the group formation and to maximize the students' skills within the formed groups. An experiment was performed with 48 students to demonstrate the ability of the approach. Experimental results indicate that our approach can optimally allocate students in an efficient way.

Key-Words: - Fairness in coalition; group formation; genetic algorithm; heterogeneous grouping.

1 Introduction

Due to the complexity of the computer software project, the well cooperation of the group's members is becoming more important because it helps reduce individuals' workloads. Therefore, the project can be done and delivered successfully on time. Moreover, working in the group makes it possible to enhance students' ability to manage and solve project efficiently. Currently, how to gain individual student achievement in groups is important [1]. In reference [2], it is presented that one principle of cooperative learning is heterogeneous grouping. Therefore, various techniques of cooperative learning have been proposed by researchers [3], [4]. Additionally, the benefit of heterogeneous grouping is to ensure equal opportunities for all students [5]. However, few studies of learning styles have considered the formation of computer science and technology students for software development projects at a university [6]. In general, the success of student groups in developing a software project depends on various factors such as the personalities, expertise, performances, and the collaboration of the people involved in the group [7]. It has been observed by some researchers that heterogeneous groups are better in a broader range of tasks [8].

In general classes, instructors can assign students to a group, or students can choose group members themselves which is called self-selection. In theory, grouping works under the assumption that groups work better when all members are balanced in terms of diversity based on educational skills or personality differences. It is claimed by Wang et.al [9] that heterogeneous groups should be comprised of students whose prior knowledge should not be unequal. Therefore, every student succeeds when the whole group succeeds. Everyone must work in the group to complete tasks efficiently. In reality, however, this task is far more complex than what it seems if the size of the class is large [10].

At the time of forming student groups, teachers can ask questions with some sorts of association rules for students to answer, so we can obtain personalities and other information directly from students. Moreover, teachers can obtain some of the student's academic record such as the grade point average (GPA), and the registered courses to help measure students for forming groups. Therefore, searching for an optimized group of students by an

exhaustive search is not practical because it is time-consuming.

In this paper, we consider the coalition formation for computer software students in developing assigned software projects based on heterogeneous grouping in which the fairness and equity of the group formation are concerned by applying the Euclidean distance[1] to all students. This will be done using genetic algorithm, in order to maximize the student's programming skills within the formed groups.

There are six parts to this paper including this introduction. The rest of this paper is organized as follows. Section 2 gives a formal definition of the problem. The genetic algorithm, which is a tool for forming student groups, is described in Section 3. Section 4 shows how we can apply the genetic algorithm to our problem. Section 5 demonstrates the experiment of our case study, a class of CS250 (Discrete Structures) in the 1st semester 2012 at Bangkok University. The conclusion and further work is illustrated in the last section.

2 Problem Formalization

Let $S = \{s_1, s_2, \ldots, s_n\}$ denote the set of *n* students. Each student has *m* attributes, which are represented in a multi-dimensional vector. We assume $A_i = (a_{i1}, a_{i2}, \ldots, a_{im})$ to be the vector of *m* attributes of the student *i*. Each of a_{jk} has different values which depend on its category values. For example, the attribute can be the cumulative grade or the grade of previous courses. As we know, students have different values of attributes. In order to achieve our goal of fairness and equity, each group will be compared with the others. When students are in the group, attributes of the particular group can be calculated by the average value of all members. The different value of attributes of two groups, which are called g_k and g_j, can be calculated by the Euclidean distance (ED) as seen in (1).

$$ED = |g_k - g_i| = \sqrt{\sum_{j=1}^{m} |A_{kj} - A_{ij}|^2} \quad (1)$$

,where A_{kj} is the value of attribute *j* of the group *k*.

We assume that students are divided into m groups denoted $G = \{g_1, g_2, \ldots, g_m\}$. Each student belongs to only one group. The choice of group size involves difficult trade-offs. According to Rau and Heyl's paper in [11], groups of three students contain less diversity; and may lack divergent thinking styles. On the other hand, in larger groups it is difficult to ensure that all members participate and learn equally. In our experiment, the size of the group is set to be equal to four. If a certain set of students is comprised of s_a, s_b, s_c, s_d, s_e, s_f, and s_g, it can be set as two groups; $g_1 = \{s_a, s_b, s_c, s_d\}$ and $g_2 = \{s_e, s_f, s_g\}$.

3 Basic Concepts of Genetic Algorithms

Genetic algorithm (GA) was introduced in the United States in the 1970s by John Holland at the University of Michigan. In particular, genetic algorithm works very well on real complex world problems. Therefore, it has become a popular methodology for a variety of complex problems [12], such as fractal image compression [13], nurse roistering problem [14], and coalition formation among buyer agents [15][16]. Also it was applied by Wang et.al. [9] to help create heterogeneous groups by psychological factors. In general, a genetic algorithm mimics the notions of natural evolution to the world of computers, and imitates natural evolution. Therefore, a particular problem must be encoded as a chromosome. The problem can be encoded in various ways with the aim of solving the problem. The fitness value of each chromosome must be calculated in an exact way associated with the chromosome's structure. A suitable fitness function is designed to help evaluate chromosomes to search for the optimum solution. While GA is running, populations of current generation are the most promising solutions for the previous generation. Then, genetic algorithm creates a population of solutions and applies three genetic operators, reproduction, mutation, and crossover operator. They are basic operators to search for the best solution(s).

4. Detail of the Genetic Algorithm for Heterogeneous Grouping (GAHG)

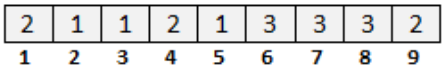
4.1 Problem Encapsulation

An important part of genetic algorithm is to encode the variables of our problem into the chromosome. The design of our chromosome is shown in Fig. 1(a). As a set of $S = \{s_1, s_2, \ldots, s_n\}$ denoted the set of *n* students, the length of chromosome is equal to n.

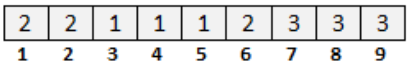
[1] Euclidean distance is also named Euclidean metric which is the "ordinary" distance between two points.

If students are divided into m different groups, the *size* of the group is $\left\lfloor \frac{n}{m} \right\rfloor \le size \le \left\lceil \frac{n}{m} \right\rceil$. Then, the value of each element of the chromosome ranks from 1 to m. For instance, if a set of 9 students is divided into 3 groups, the size of each group is $\left\lfloor \frac{9}{3} \right\rfloor = 3$. Suppose that they are $g_1=\{s_2, s_3, s_5\}$, $g_2=\{s_1, s_4, s_9\}$, and $g_3=\{s_6, s_7, s_8\}$. The chromosome can be encoded as in Fig. 1(b), named chromosome$_x$. The other called chromosome$_y$. Fig. 1(c) represents that there are three groups of $g_1=\{s_3, s_4, s_5\}$, $g_2=\{s_1, s_2, s_6\}$, and $g_3=\{s_7, s_8, s_9\}$ respectively.

(a) Structure of chromosome



(b) chromosome$_x$



(c) chromosome$_y$

Fig. 1 The chromosome structure for student group formation and examples

4.2 Fitness Function

As assumed earlier, student s_i is associated with the multi-attributes vector $A_i = (a_{i1}, a_{i2}, \ldots, a_{ip})$, where *p* is the maximum number of attributes. The average value of all members in the same group must be calculated, as it is the attribute of the group. In this paper, we apply the cumulative GPA and the grades of prior programming courses as the student's attributes because they represent the student's performance and educational skill. Suppose 2-attribute vector of student i is $A_i = (a_{i1}, a_{i2})$ and all vectors of nine students are presented in Fig. 2. Then, attributes of each group associated with a certain chromosome are calculated by the average value of all members represented in Table 1.

A_1= (2.50,C), A_2= (2.14,D), A_3= (3.50,B), A_4= (2.40,B+), A_5= (2.50,C), A_6= (3.14,A), A_7= (2.19,C), A_8= (3.40,A), A_9= (1.98,C+)

Fig. 2 Attributes of nine students

For chromosome$_x$, since student s_2, s_3, and s_5 are in g_1, the average value of first attribute (A_{11}) is

$$a_{11} = \frac{2.12 + 3.50 + 2.50}{3} = 2.71 \cdot$$

And the average value of second attribute (A_{21}) is

$$a_{12} = \frac{D + B + C}{3} = \frac{1 + 3 + 2}{3} = 2.0 \cdot$$

, where A=4, B+=3.5, B=3, …, D+=1.5, D=1, F=0, and others, such as W (Withdrawal) and I (Incomplete), are set to be zero.

Consequently, the vector attribute of g_1 = (a_{11}, a_{12}) is (2.71, 2.0). With the same calculation, the vector (a_{21}, a_{22}) of $g_2=\{s_2, s_3, s_5\}$ can be shown as

$$a_{21} = \frac{2.50 + 2.40 + 1.98}{3} = 2.29 \cdot$$

And the value of second attribute (A_{22}) is

$$a_{22} = \frac{C + B^+ + C^+}{3} = \frac{2 + 3.5 + 2.5}{3} = \frac{8}{3} = 2.67 \cdot$$

Then, we apply the calculation to the other groups. The vector attribute of g_2 represented as (a_{21}, a_{22}) is (2.29, 2.67). Also, the vector attribute of g_3 is (a_{31}, a_{32}) = (2.91, 3.33).

Based on fairness and equity among students, all formed groups should be comprised of students with similar levels of performance. Therefore, each group will be compared with the others. Fitness values of each chromosome can be calculated by using p-dimensional Euclidean space represented as follows:

$$fitness(chromosome_x) = \sum_{k=1}^{m-1} \sum_{j=k+1}^{m} \sqrt{\sum_{i=1}^{p} \left| a_{ki} - a_{ji} \right|^2} \qquad (2)$$

, where m is the number of groups, p is the number of attributes, and a_{ki} is the value of attribute i of the group *k*.

The calculation of fitness value for chromosome$_x$ and chromosome$_y$ are shown below.

ISBN: 978-1-61804-140-1

$$fitness(chromosome_x) = \sum_{k=1}^{2}\sum_{j=k+1}^{3}\sqrt{\sum_{i=1}^{2}\left|a_{ki}-a_{ji}\right|^2}$$
$$= \sqrt{\left|a_{11}-a_{21}\right|^2+\left|a_{12}-a_{22}\right|^2} + \sqrt{\left|a_{11}-a_{31}\right|^2+\left|a_{12}-a_{32}\right|^2} + \sqrt{\left|a_{21}-a_{31}\right|^2+\left|a_{22}-a_{32}\right|^2}$$
$$= \sqrt{\left|2.71-2.29\right|^2+\left|2.0-2.67\right|^2} + \sqrt{\left|2.71-2.91\right|^2+\left|2.0-3.33\right|^2} + \sqrt{\left|2.29-2.91\right|^2+\left|2.67-3.33\right|^2}$$
$$= \sqrt{0.42^2+0.67^2} + \sqrt{0.2^2+1.33^2} + \sqrt{0.62^2+0.66^2}$$
$$= 0.790759 + 1.344954 + 0.9055385 \approx 3.0412515$$

$$fitness(chromosome_y) = \sqrt{\left|2.8-2.59\right|^2+\left|3.5-2.33\right|^2} + \sqrt{\left|2.8-2.5\right|^2+\left|3.5-2.83\right|^2} + \sqrt{\left|2.59-2.5\right|^2+\left|2.33-2.83\right|^2}$$
$$= \sqrt{0.21^2+1.17^2} + \sqrt{0.3^2+0.67^2} + \sqrt{0.09^2+0.5^2}$$
$$= 1.188697 + 0.734098 + 0.5080354 \approx 2.4308304$$

In our algorithm, we need chromosomes with low fitness values. The higher fitness values mean that the chromosome is bad in distributing students among groups. Weak performing students may have chosen to be together in the same group as well as good performing students. On the other hand, the lower fitness values show that the chromosome arranges different students into formed groups properly. Low performing students will have little chance to be together and high performing students are assigned to be in difference groups. Therefore, in this example $chromosome_y$ is better than $chromosome_x$ in forming student groups.

4.3 Operators for Generating Offspring

Once the chromosome structure and fitness function are completely constructed, genetic operations must be designed to perform within a single generation. The purpose of these operators is to create various new solution vectors that are shown to be good. Note that in our paper the algorithm works on fixed-length character strings. If the number of students is *n*, the fixed-length of character strings for our chromosome is equal to *n* as well. In our algorithm, reproduction operation is the first operator that applies to copying existing population in the current generation to the next generation. After that, the following operator is a crossover operation. The operator helps maintain diversity in the population, as crossover causes the chromosomes of offspring to be different from their parents. For our algorithm, we use two types of crossover techniques which are one-point crossover and self-crossover. The one-point operator creates two new offspring from two existing parents by randomly interchanging substrings at the random part. The example of one-point crossover operation, which applies on $chromosome_x$ and $chromosome_y$, is depicted in Fig. 3. In our crossover process, the small value called crossover probability p_c is set to preserve some of the good strings that are already present in the current generation. As the crossover operator is mainly responsible for the search for new offspring, it might be possible to produce a bad individual. However, these offspring will never be selected for next operations based on fitness function. Self-crossover switches values of two elements of the selected chromosome. This technique helps the algorithm maintain the size of all groups, since our algorithm strongly requires the number of students in most groups to be as equal as possible. The example of self-crossover technique which applies on $chromosome_x$ is illustrated in Fig. 4. One cycle of these operations, reproduction operator and crossover operator, applies on the current population to create a new population in a single generation. This new population obtained from the previous generation is used to generate subsequent populations and so on. Finally, the algorithm yields the best solution that is closest to the optimum solution.

TABLE 1 ATTRIBUTE VALUE OF EACH GROUP ON DIFFERENT CHROMOSOME

	Attribute Values of Each Group		
chromosome	**Group1 (g1)**	**Group2 (g2)**	**Group3 (g3)**
$chromosome_x$	$(2.71, 2.00) = \begin{cases} A_2 = (2.12, D) \\ A_3 = (3.50, B) \\ A_5 = (2.50, C) \end{cases}$	$(2.29, 2.00) = \begin{cases} A_1 = (2.50, C) \\ A_4 = (2.40, B+) \\ A_9 = (1.98, C+) \end{cases}$	$(2.91, 3.33) = \begin{cases} A_6 = (3.14, A) \\ A_7 = (2.19, C) \\ A_8 = (3.40, A) \end{cases}$
$chromosome_y$	$(2.80, 3.50) = \begin{cases} A_3 = (3.50, B) \\ A_4 = (2.40, B+) \\ A_5 = (2.50, C) \end{cases}$	$(2.59, 2.33) = \begin{cases} A_1 = (2.50, D) \\ A_2 = (2.14, D) \\ A_6 = (3.14, A) \end{cases}$	$(2.50, 2.83) = \begin{cases} A_7 = (2.19, C) \\ A_8 = (3.40, A) \\ A_9 = (1.98, C+) \end{cases}$

ISBN: 978-1-61804-140-1

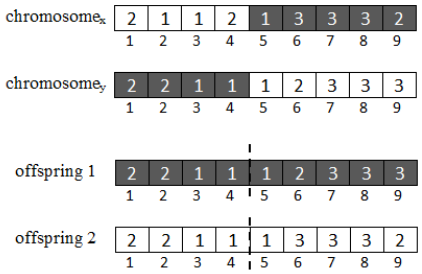


Fig. 3 One-point crossover operation

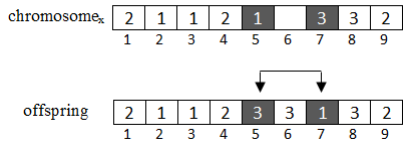


Fig. 4 Self-crossover operation

5. Experiment and Results

To confirm the efficiency of our algorithm, we conducted an experiment of 48 students enrolled in CS250 for 1st semester/year 2012 at Bangkok University as a case study. In constructing our GAHG algorithm, specific parameters, such as initial population size (M), number of generations (Gen), and crossover probability (p_c) are quite important, as they can greatly influence the performance of the algorithm. Therefore, we had tried several runs with different values of parameters to see which values would yield the best solution. Finally, the value of initial population size (M) is 300, number of generations (Gen) equals 100, and crossover probability (p_c) is 0.21. As the set of 48 students registered in CS250 was adopted in our experiment, we decided to form a small group of four students. Thus, there are 12 groups of four students each. With this number of groups we are able to follow student progress and software projects efficiently. As the algorithm aims to balance students' skills among formed groups, our algorithm would not set a group of another size unless it is impossible to do that. The results of our proposed algorithm are illustrated in Fig. 4 – Fig. 7. The graph of Fig. 4 presents that the average cumulative grade point (GPA) of all groups, are quite similar, which is about 2.67. It implies that students with low performance and students with high performance are balanced in an efficient way. Good students are optimally distributed to be in different groups, while there is no group formed by low performing students. Furthermore, we not only balanced students by using student cumulative grade point average, we also formed student groups with the prerequisite courses for CS250. Based on the curriculum of computer science at Bangkok University, two prior courses for CS250 are CS311 Computer programming II and CS317 Visual Programming. The average grade of first course of the formed groups is 2.88, and the second prerequisite course is 2.77. As illustrated in both Fig. 6 and Fig. 7, in terms of heterogeneity, students with different grades are optimally distributed among groups making the average grade of each prerequisite course for all groups quite similar.

6 Conclusions and Further works

In this paper, an approach called GAHG is proposed by using a genetic algorithm to help us generate the student groups based on heterogeneous grouping. We aim to develop the method to apply to computer science and technology students at Bangkok University in developing software project. A student will be assigned to a group by student's educational skills in which both fairness and equity in terms of learning performance and programming skills are strongly concerned. Based on our case study and experimental results, it shows that our approach can optimally allocate heterogeneous students, supported by the graph presented in Fig. 5-Fig.7. As a result of algorithm, weak performing students will have little chance to be together in the same group. On the other hand, good students will be assigned in difference groups. By doing this, all established groups will be more balanced. Additionally, we hope that by assigning students to the right group it is possible to upgrade their abilities by learning from others. Therefore, the groups may be eligible to develop complex software projects and the whole students can manage their computer software projects efficiently. As a result, the productive outcome of these groups may be enhanced.

In the future, the result of our algorithm will be compared to other algorithms and optimization methods. Of course, the algorithm will be compared to the self-selection method made by students or teachers. And then, at the end of a semester, the knowledge and programming skills that dissimilar students received after joining in generated groups will be evaluated to see the efficiency of the algorithm. We also plan to apply the algorithm to other datasets to see the scalability of the approach. Some other attitudes and student preferences will be included in the fitness function. Multiple objectives along with the Pareto-optimal function will be applied to achieve more fairness and equity in groups.

ISBN: 978-1-61804-140-1

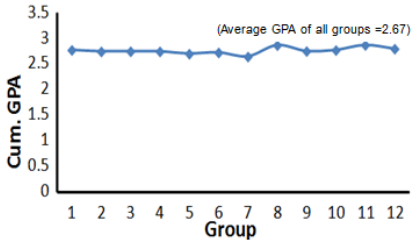


Fig. 5 Cumulative grade point average (GPA)

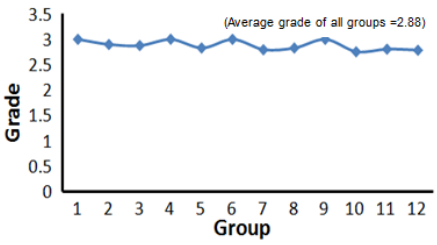


Fig. 6 Average grade of CS311

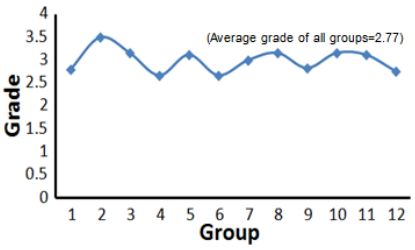


Fig. 7 Average grade of CS317

References:

[1] Renner, G., & Ekárt, A. (2003). Genetic algorithms in computer aided design. Computer-Aided Design, Elsevier Science Ltd., vol. 35, Issue 8, July 2003, pp. 709-726.

[2] Pedro, P., Alvaro, O., & Pilar, R. (2010). *A Method for Supporting Heterogeneous-Group Formation through Heuristics and Visualization.* Journal of Universal Computer Science, vol. 16, no. 19 (2010), pp. 2882-2901.

[3] Wessner, M., & Pfister, H. (2001). Group Formation in Computer-Supported Collaborative Learning. Proceedings of the 2001 International ACM SIGGROUP Conference on Supporting Group Work Boulder.

[4] Oldfield, S., & Morse, D. (2005). Truly Virtual Teams: (Team) Working-In-Progress. Proceedings of the 6th Annual conference of the Subject Centre for Information and Computer Sciences, pp. 30-33.

[5] Sapon-Shevin, M., Ayres, B., & Duncan, J. (2002). Cooperative learning and inclusion. In J.S. Thousand, R.A. Villa, & A.I. Nevin (Eds.), *Creativity and collaborative learning: The practical guide to empowering students, teachers, and families (2nd ed.)*, Baltimore: Paul H. Brookes, pp. 209-222.

[6] Gilbert, J. E., & Swanier, C. A. (2008). Learning Styles: How Do They Fluctuate? Institute for Learning Styles Journal, vol. 1, Fall 2008, pp. 29-40.

[7] Bekele, R. (2005). Computer-Assisted Learner Group Formation Based on Personality Traits, PhD thesis. Universität Hamburg, Hamburg.

[8] Martin, E., & Paredes, P. (2004). Using learning styles for dynamic group formation in adaptive collaborative hypermedia systems. The First International Workshop on Adaptive Hypermedia and Collaborative Web-based Systems (AHCW 2004), pp. 188-198.

[9] Wang, D.-Y., Sunny, S. L., & Chuen-Tsai, S. (2003). DIANA: A computer supported heterogeneous grouping system for teachers to conduct successful small learning groups. Computers in Human Behavior, 23 (2007), pp. 1997–2010.

[10] Harrison, G., Griffin, S., & Broughton, L. (2009). The Innovation Bangkok: Project Group Formation and Much More. 10th Annual Conference of the Subject Centre for Information and Computer Sciences, pp. 44-49.

[11] Rau, W. and Heyl, B. S. (1990). Humanizing the college classroom: Collaborative learning and social organization among students. Teaching Sociology, 18, pp.141-155.

[12] JC Chen, CC Wu, CW Chen and KH Chen (2012). Flexible job shop scheduling with parallel machines using Genetic Algorithm and Grouping Genetic Algorithm, Expert Systems with Applications, Vol.39, Issue 11, 1 September 2012, Pages 10016–10021.

[13] Xing-yuan, W., Fan-ping, L., & Shu-guo, W. (2009). *Fractal image compression based on spatial correlation and hybrid genetic algorithm.* Journal of Visual Communication and Image Representation genetic algorithm, pp. 505-510.

[14] Z. Bäumelt, P. Šucha and Z. Hanzálek (2011). A Genetic Algorithm for a Nurse Rerostering Problem, the 10th Workshop on Models and Algorithms for Planning and Scheduling Problems, Czech Republic. pp.70-78.

[15] Hyodo, M., Matsuo, T., & Ito, T. (2003). An Optimal Coalition Formation among Buyer Agents based on a Genetic Algorithm. International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE'03), Loughborough, UK, pp. 759–767.

[16] Boongasame, L. and Sukstrienwong, A. (2009). Buyer coalitions with bundles of items by using genetic algorithm. Emerging Intelligent Computing Technology and Applications Lecture Notes in Computer Science, 2009, Volume 5754/2009, pp. 674-685.