ABSTRACT

WANG, YINGHUI.. Dynamic Assignment of Peer Reviewers for Teams (Under the direction of Dr. Edward F. Gehringer.)

Team peer review is a superset of individual review. We can consider individual peer review as team peer review with only one student in each team. Although much work has been done on peer review in an academic setting, little work has been done on the strategies for mapping reviewers, especially in team peer review. In the usual classroom setting, reviewers are mapped statically and randomly, e.g., instructors collect submissions, shuffle them, and pass them back in the same class or next class, The situation is more complicated when peer review is electronic and asynchronous. If the mapping strategy is static, there is no guarantee that each reviewer assignment is *valid.* A student might drop the course during the assignment period, and thus be unable to complete the reviews (s)he is assigned. Or (s)he might not submit, leaving his reviewers with nothing to review. Therefore, in this case some students would review more, and some would review less than others.

In this work, two dynamic strategies are developed for team peer review. These strategies are adapted to situations like students dropping the course or not doing their assigned reviews when changed situations can be updated in time. In the first strategy, each student is required to do the same number of reviews, and the number of reviewers per team is balanced as nearly as possible. In the second strategy, each team is required to have the same number of reviewers, and the number of reviews per student is balanced as nearly as possible.

DYNAMIC ASSIGNMENT OF PEER REVIEWERS FOR TEAMS

By

YINGHUI WANG

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

Raleigh
2002

Approved by:

_____ _____

_____
Chairman of Advisory committee

DEDICATION

I would like to sincerely dedicate this thesis to my husband, Xiaobo, my mom and brothers and sister. I'm so proud and lucky to have them as my family. They are always there to love me and support me.

BIOGRAPHY

Yinghui Wang was born in Shandong, China. Having achieved her undergraduate degree in Chemistry from Shandong University, China, she began her graduate studies in the University of Alabama in Huntsville. She majored in Chemistry, then in Computer Science, in the meantime working as a teaching assistant. She transferred to North Carolina State University in January of 2000, and began to pursue her master degree in Computer Engineering. She has studied a number of outstanding courses and worked on some interesting projects on Algorithm Design and Analysis, Java Programming, Object-Oriented Techniques, Computer Networks, and Database. In the summer of 2000, she joined in IBM as a Co-op software engineer. She worked on software development and testing for Tivoli Internet Services Manager. The one year's working experience in IBM has rewarded her much in the capabilities of both software development and team corporation. She would like to continue the computer-related research and development work after graduation.

ACKNOWLEDGMENTS

Words cannot express the gratitude and indebtedness I feel towards Dr. Edward Gehringer. He has been my mentor throughout the past year, and has always given me so much of advice, support and friendship. Without him, this thesis and research work would have been impossible. Thank you so much Dr. Gehringer.

I am also grateful to Dr. Matthias F. Stallmann and Dr. George H. Wahl for their kindness to be my committee members, their assistance and help.

I would like to sincerely thank my team in IBM: these include Toni Sweetland, David Houck, Vindvashni P Tiwari, Steven M Krol, Karen Chaney, and Nathan Baker I am grateful to them for their corporation, support and helps when we were working together on such a great project.

Last but not the least, I wish to acknowledge the help of Gopal R Srinivasa and Prabhas Ranjan Sinha, who provided information on the Peer Grading System.

TABLE OF CONTENTS

TABLE OF FIGURES

CHAPTER 1: INTRODUCTION

1.1. Introduction to Peer Review Mapping Strategies

Peer review in the classroom is becoming an increasingly important technique. In the last ten years, much work has been performed on accessing usefulness of the technique (students generally like it, and learn well from it) and its validity (students do in general rate better work more highly, though some effort needs to be invested in the assessment procedure to assure this). There have been over 100 papers published [GC 02]. However, few published reports discuss appropriate strategies for matching reviewers with reviewees [Gehr 01].

In his 1998 survey paper [Topp 98], Topping says, "How peer assessors and assessees should best be matched ... is discussed surprisingly little in the literature." In most cases, he says, a single assessor was matched with an assessee [GC 02]. In other cases, multiple assessors were used. The matching has been done along two dimensions: blindly or non-blindly, electronically or non-electronically [Gehr 01].

Often, reviews are done blindly, e.g., by collecting student assignments in one class, and passing them out to other students in the next class period, using an instructor-assigned ID number to identify the students [KPD 95]. However, some projects use face-to-face interaction, frequently called "peer revision" [Sty98, Sty 99]. In this case, of course, review is not blind. In cases where electronic review is done, usually the review is done blindly [Gehr 01], via an application such as the Daedalus Integrated Writing Environment [Daed 97], but sometimes it is done non-blindly by e-mail [DB 97].

Most papers ignore any indication of how reviewers and the reviewees are matched; others just say they are matched "randomly." Random matching means that each time the mapping is done, a different result (set of reviewee-reviewer pairings) will be produced [Gehr 01]. Often, random matching is easy to do. In the last example I mentioned, in one class, if papers are collected and passed out to other students, each student will get a paper to review, and each student will be reviewed by another student. However there is no such guarantee when collected papers are passed out in the next class. Some students who are assigned to do review will not do them, either because they just don't do their homework, or because they drop the course [GC 02]. Some students will not be reviewed by their classmates because their reviewer drops the course. This is a major problem in a large class, and happens both to individual review and team review.

From the above statement, we can see the problems with "static" strategies of mapping reviewers to reviewees, which map reviewers with their reviewees before the review process begins. These problems can be solved if we use "dynamic" strategies, which do not assign review work to a student until the student asks to do a review. Using a dynamic strategy, no student will be assigned to review work that has not been submitted. Conversely, a student who has submitted the work will be assigned a "live" reviewer [GC 02].

Normally, when assignments are done by individuals, other individuals are assigned to review them. When projects are done by teams, they may also be reviewed by

individuals; let us call this a *team peer-review* strategy. In this case, students are assigned some number of other teams to review. Actually, individual peer review can be considered to be a subset of team peer review, viz., review of one-member teams.

1.2. Thesis Outline

As described above, team peer review is a superset of individual peer review. Effective strategies for dynamic team peer review are the topic of this thesis.

In Chapter 2, the team peer review mapping strategy is introduced, by giving two algorithms: an algorithm where every student reviews the same number of teams (the "ERee" algorithm), and an algorithm where each team has the same number of reviewers (the "ERer" algorithm). Chapter 3 describes the implementation and analyzes the performance of the ERee algorithm and ERer algorithm. Chapter 4 is the conclusion and future work.

CHAPTER 2: A TEAM PEER REVIEW STRATEGY

Teamwork is popular in classes, as is peer review. Therefore, team review mapping plays an important role. Strategies for team peer review are an extension of strategies for individual peer review. In individual peer review, each student has his/her own submission, and the number of reviewers is equal to the number of reviewees. In team peer review, the numbers are different. Therefore compared with individual peer review, there are more factors to be considered, and it's more difficult to design a team peer-review strategy.

2.1. Introduction to Team Peer Review Strategy

Team peer review is used in an environment where students work in teams. "Mapping" a reviewer means assigning one student to review one "team." A constraint on this strategy is that no student will review his/her own team.

In the case of individual peer review, if each author is assigned *r* reviewers, then each reviewer will have *r* authors to review. When multi-member teams are being reviewed, the number of reviews a team gets is *not* equal to the number of reviews done by each reviewer. In fact, depending on the size of the teams and the number of reviewers, it may not be possible both to have each reviewer review the same number of teams *and* have each team reviewed by the same number of reviewers. For example, in a class with 30 students working in 7 teams, if we want each student to have 3 teams to review, the number of reviewers each team has is (30×3/7), which is not an integer. That means each team can't have the same number of reviewers. Similarly, if each team is

required to be reviewed by 8 students, the number of reviews done by each student is (7×8/30), which means each student needs to review one or two teams.

Thus, with team review, we have two algorithms for these situations. When each reviewer is required to review the *same number of teams*, the ERee (equal number of reviewees) algorithm is used. On the other hand, when each team needs to get the *same number of reviewers*, it's time for the ERer (equal number of reviewers) algorithm. These two algorithms are explained in the next section of this chapter. The following two definitions are suitable to both of the algorithms.

Definition 1. A mapping assignment of a reviewer to a team is *valid* if it does not cause a reviewer to review his/her own team, and leaves sufficient valid mapping assignments for all future reviewers.

Definition 2. A mapping assignment is *invalid* if it does not leave enough valid mapping assignments for all future reviewers, or it causes the reviewer to review his/her own team.

If previous mapping assignments are all valid, we are guaranteed to be able assign a valid mapping to the next reviewer, and so forth, until all reviewers have been mapped to the appropriate number of teams to review.

2.2.EREe Algorithm

2.2.1. Outline of EREe Algorithm

As we said on the above, EREe stands for equal number of reviewees, which means that each reviewer will review the same number of teams.

Let's assume that in one class, the number of students is *n*, the number of teams is *t*, and each student is required to review *r* teams. Therefore, *t* should be less or equal to *n*, and we also assume *r* is less or equal to *t* (that's because the reviewer can't review his/her own team).

A matrix is used in EREe Algorithm. In this matrix, each row is for a particular reviewer, and each column is for a particular team (submission). Let's call this matrix *team_review*[1:*n*][1:*t*]. The status of one mapping assignment is shown by the corresponding value in this matrix. A valid assignment is represented by "1". An invalid assignment is represented by "0", and "–1" means a reviewer is not be mapped to this assignment yet. For example,

- *team_review*[*i*][*j*] = 1 means that team *j*'s submission is assigned to student *i*.
- *team_review*[*i*][*j*] = 0 means that team *j*'s submission cannot be reviewed by student *i*, and
- *team_review*[*i*][*j*] = –1 means that team *j*'s submission can be assigned to student *i* to review, but we have not made this mapping assignment yet.

Definition 3.1. A mapping assignment of a reviewer to a team is *valid* in the ERee algorithm if each team is reviewed by the same number of individuals, as nearly as possible.

To meet the requirement of Definition 3.1, three numbers are needed, the maximum number of reviewers that one team can have (*max*), the minimum number of reviewers that one team can have (*min*), and the number of teams (*num_max*) which have *max* reviewers. All of these three variables are calculated by the algorithm. To make each team be reviewed by the same number of individuals, as nearly as possible, we assume *max* is equal to *min* if *num_max* is equal to 0; otherwise, *max* is 1 larger than *min*.

The basic logic of this algorithm is to prevent—

- the number of "1"s from becoming larger than *r* in any row, and
- the number of "1"s in any column becoming greater than *max*;

that is, to prevent the mapping from producing an invalid assignment. Similarly, the number of "0"s cannot be greater than ($t - r$) in any row, and the number of "0"s in any column can't be greater than ($n - min$).

Definition 3.2. In the ERee Algorithm, one row/column has **enough** "1"s if the number of "1"s in this row/column has reached the limit. Each row's limit is *r*, and each column's limit is *max* if the number of teams, which have *max* reviewers, is smaller than *num_max*; otherwise, this number is *min*.

We know in ERee Algorithm, each student needs to review *r* teams, so the limit of each row is *r*. In each column, if there are already enough teams (*num_max teams*) that have *max* reviewers, this team can only have *min* reviewers; that is, there will be, *min* "1"s in this column.

Definition 3.3. In the ERee Algorithm, one row has **enough** "0"s if number of zeros in this row has reached ($t - r$), and one column has **enough** "0"s if number of "0"s in this column has reached ($n - max$), or ($n - min$) when max reviewers is not acceptable for this team (i.e., there are already *num_max* teams that have max reviewers).

In ERee algorithm, each student should review *r* teams (there should be *r* "1"s in each row). Therefore, the number of "0"s in each row should be ($t - r$). From definition 3.3, we know there should be *max* or *min* "1"s in each column, and the number of "0"s in this column should be ($n - max$) or ($n - min$).

If any row or column has **enough** "1"s, no more mapping assignments are assigned to this reviewer or team, so the remaining elements in this row or column that are "–1" should now be set to "0". Similarly, if there are **enough** "0"s in any row or column, the remaining elements in the same row or column that are "–1" should now be set to "1".

If an element is changed in a matrix, both of its corresponding row and column are affected. Therefore, after changing a value in a row, we need to check the corresponding column and vice versa to make sure this change doesn't create an invalid value in another

row or column. So, after this algorithm sets a value in a row/column, it checks the corresponding column/row and makes a change if necessary. If the algorithm cannot make the required change without violating some other constraint, it undoes changes already made pursuant to this mapping assignment, and chooses another of the possible mapping assignments.

Since students cannot review their own team, at the beginning, we set each element representing a student and his/her own team "0". For example, if in a class student i works in team t_i, we set $team\_review[i][t_i] = 0$. Other elements in this matrix are all initialized to "-1".

Let's use t_i to represent the team of the i^{th} student.

Definition 4. A mapping is valid if no row or column has too many ones "1"s, and no row or column has too many "0"s, and if $team\_review[i][\ t_i] = 0$ (for $i = 1, 2, \ldots, n$).

We will prove that there is a legal mapping assignment for the i^{th} student, assuming that the $(i–1)^{st}$ student's mapping assignment has been done successfully ($i = 2,\ldots, n–1, n$).

When student i asks to do a review, we search the i^{th} row and select an element whose value is "–1". Then we check whether the submission corresponding to this element has been made; if so, we set the value of this element to "1"; if not, we select

another element. Let's say we select *team_review*[*i*][*j*]. There are two types of check we will do after we set *team_review*[*i*][*j*] = 1.

- First, we count the number of "1"s in this row. If this row has **enough** "1"s, we need to set all the remaining "–1" elements to "0"s (this student has enough reviewees, so no more reviewees are needed). For each "–1" element (say, in column *j′*) set to 0 in this row, we check the j^{th} column. Since the number of "1"s has not been changed, we only check the number of "0"s in the j^{th} column. If it has **enough** "0"s, the remaining elements with value "–1" in the j^{th} column will be set to "1" (all of the remaining students are set to review j^{th} team, or this team won't have sufficient reviewers).

- The second type of check is to check the number of "1"s in the j^{th} column, since we have set *review*[*i*][*j*] = 1. If the number of "1"s in the j^{th} column is not **enough**, it is valid. If the j^{th} column has enough "1"s already (this means the j^{th} team has enough reviewers), further action is needed. First, since no more reviewers are needed for this team *j*, we need to change all the remaining "–1"s in the j^{th} column to "0"s. Let's say the row number of such an element is *i′*. Then, we check i^{th} row to see whether it is still a valid mapping assignment. Here only the number of "0"s is changed. So we check whether the number of "0"s has exceeded (*t* – *r*).

Before an element is changed, we need to back up its corresponding row and column in case an invalid mapping assignment is attempted. If all of the above checks are valid, a

successful mapping assignment, *review*[*i*][*j*] = 1, is made. If any of the above checks fails, then our current mapping assignment, *team_review*[*i*][*j*] = 1, is invalid. What we need to do is to set *review*[*i*][*j*] = 0, and restore all the changed elements to "–1" and try another element in the i^{th} row.

The following section is an example, which shows how this process takes place.

2.2.2. Example: How ERee Algorithm works

Suppose there are six students in one class, S_1, S_2, S_3, S_4, S_5, and S_6. Each student is assigned two reviews. These six students are in four teams, T_1, T_2, T_3, and T_4. $T_1 = \{S_1, S_2\}$; $T_2 = \{S_3, S_4\}$; $T_3 = \{S_5\}$; $T_3 = \{S_6\}$.

That is, n=6, r=2, t=4, and min=$n \times r/t$=3=max. We assume that all submissions are valid and we assign one review to a student when that particular student requests to do a review.

We assume that students request to do reviews in the sequence 0, 1, 2, 3, 4, 5, that is, student 0 requests to do the review first, student 1 requests second, student 2 requests third and so on.

Initially, the matrix *review* is as shown at the right.

	team_review (teams being reviewed)				*row_zeros*	*row_ones*
Student doing reviewing						
	0	–1	–1	–1	1	0
	0	–1	–1	–1	1	0
	–1	0	–1	–1	1	0
	–1	0	–1	–1	1	0
	–1	–1	0	–1	1	0
	–1	–1	–1	0	1	0
col_zeros	2	2	1	1		
col_ones	0	0	0	0		

	team_review (teams being reviewed)				*row_zeros*	*row_ones*
Student doing reviewing						
	0	**1**	–1	–1	1	**1**
	0	–1	–1	–1	1	0
	–1	0	–1	–1	1	0
	–1	0	–1	–1	1	0
	–1	–1	0	–1	1	0
	–1	–1	–1	0	1	0
col_zeros	2	2	1	1		
col_ones	0	**1**	0	0		

First request: When student 0 requests to do a review, we randomly pick one of the –1s in row 0 of the *review* matrix. Let's say we pick *team_review*[0][1]. This assigns student 0 to review team 1. Having done this, we set *team_review*[0][1] = 1.

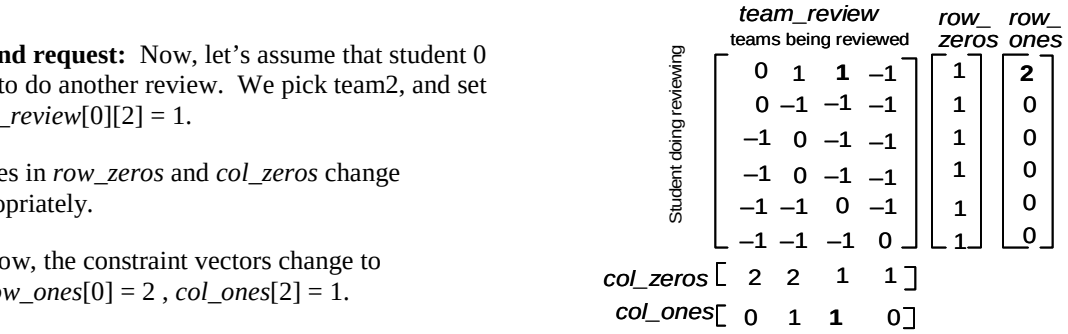
Values in *row_zeros* and *col_zeros* change appropriately.

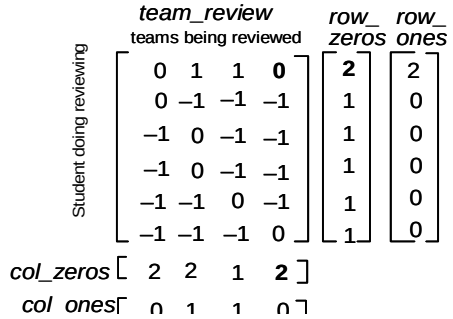
In the diagram to the left, values that have changed are shown in **boldface**.

Second request: Now, let's assume that student 0 asks to do another review. We pick team2, and set *team_review*[0][2] = 1.

Values in *row_zeros* and *col_zeros* change appropriately.

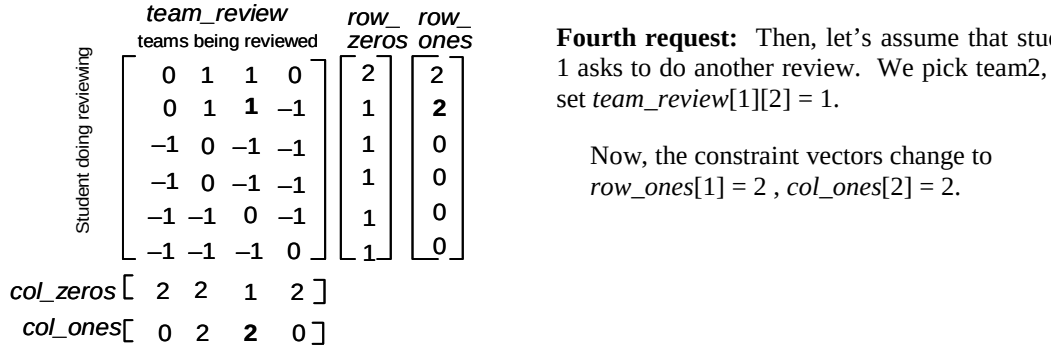
Now, the constraint vectors change to *row_ones*[0] = 2 , *col_ones*[2] = 1.





Since *row_ones*[0] = $t - r = 2$, student 0 cannot review anyone other than his current reviewees, and thus *team_review*[0][3] = 0 is set.

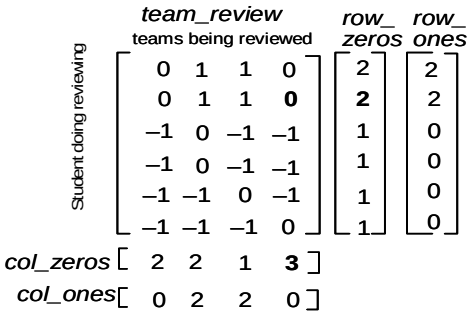
Third request: Now, it's the turn of student 1. We pick team 1 and set *team_review*[1][1] = 1.



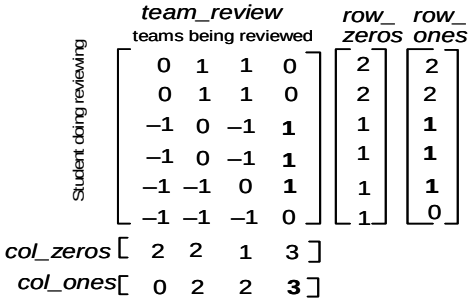
Fourth request: Then, let's assume that student 1 asks to do another review. We pick team2, and set *team_review*[1][2] = 1.

Now, the constraint vectors change to *row_ones*[1] = 2 , *col_ones*[2] = 2.

Since *row_ones*[1] = $t - r$ = 2, student 1 cannot review anyone other than his current reviewees, and thus *team_review*[1][3] = 0 is set.



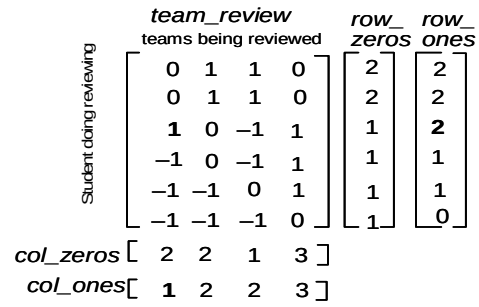
Now, another boundary condition is met, that is *col_zeros*[3] = 3. Three of the six students are now ineligible to review team 3, so the other three students must review this student, according to *min*=*max*=3. We set *team_review*[2][3] = 1, *team_review*[3][3] = 1, and *team_review*[4][3] = 1.

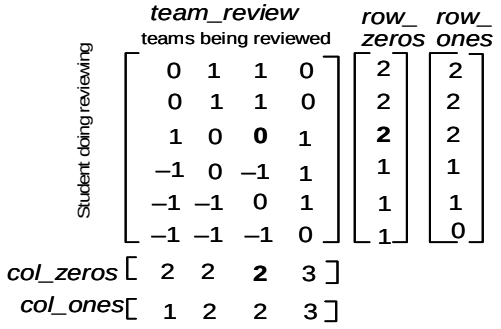


Fifth request: Now, it's the turn of student 2. We pick team 0 and set *team_review*[2][0] = 1

Now, the constraint vectors change to *row_ones*[2] = 2 , *col_ones*[0] = 1.

.



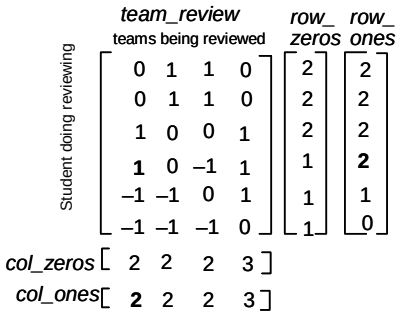


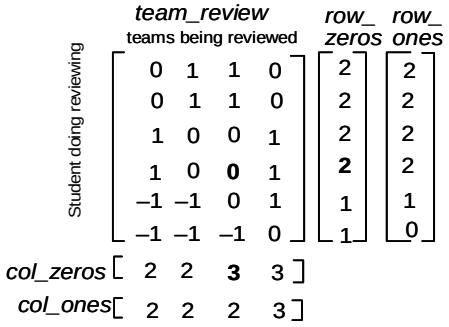
Therefore we can set the other elements with value of "–1" to "0" in row 2, because *row_ones*[2] = 2 = *r*.

That is, *team_review*[2][2] = 0

Sixth request: Now, student 3 asks to do one review. We pick team0, and set *team_review*[3][0] = 1.

Now, the constraint vectors change to *row_ones*[3] = 2 , *col_ones*[0] = 2.





Therefore we can set the other elements with value of "–1" to "0" in row 3, because *row_ones*[3] = 2 = *r*.

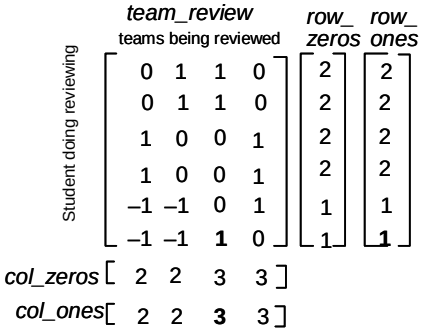
That is, *team_review*[3][2] = 0

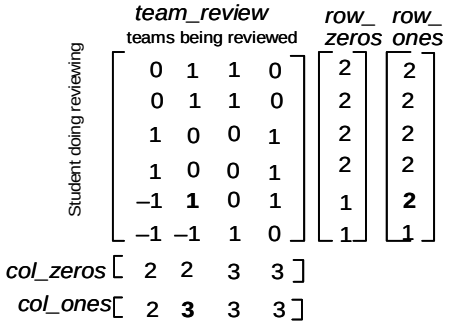
Now, the constraint vectors change to *row_zeros*[3] = 2 , *col_zeros*[2] = 3

We can see that *col_zeros*[2] = 3 = *t* – *max*, so we can set the other elements with value of "–1" to "1" in column 2.

That is, *team_review*[5][2] = 1

Now, the constraint vectors change to *row_ones*[5] = 1 , *col_ones*[2] = 3



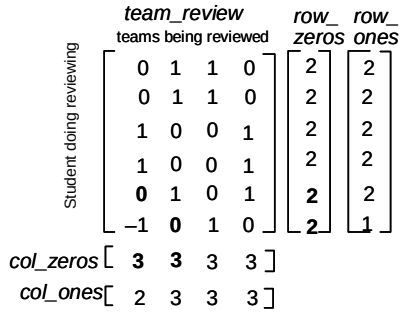


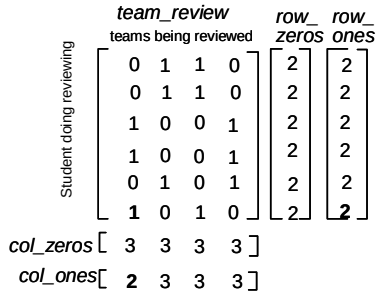
Seventh request: Now, student 4 asks to do one review. We pick team1, and set *team_review*[4][1] = 1.

Now, the constraint vectors change to *row_ones*[4] = 2 , *col_ones*[1] = 3.

Then we can set the all elements with value of "–1" in row 4 to "0", because *row_ones*[4] = 2 = *r, and* set the all elements with value of "–1" in column 1 to "0", because *col_ones*[1] = 3 = *min*.

Now, the constraint vectors change to *row_zeros*[4] = 2, *col_zeros*[0] = 3, *row_zeros*[5] = 2, *col_zeros*[1] = 3





Now, let's see the only element whose value is "–1", *team_review*[5][0]. Because *col_zeros*[0] = 3, this algorithm sets *team_review*[5][0] = 1.

From this example, we can see in a class with six students, when each student is required to have two reviews, only seven assignment steps are needed. Some reviews are assigned to students because they are the only choices.

2.2.3. ERee Algorithm

In this section is the realization of ERee Algorithm outlined in section 3.2.1. First of all, let's see the list of important variables.

- *team_review*[][]: the basic matrix used to hold the mapping;
- *row_zeros*[]: keeps track of the number of "0"s in each row;

- *row_ones*[]: keeps track of the number of “1”s in each row;
- *col_zeros*[]: keeps track of the number of “0”s in each column;
- *col_ones*[]: keeps track of the number of “1”s in each column ;
- *cells_changed*[]:keeps track of elements changed during one iteration (each element of *cells_changed* is a structure to keep track of the position of the original element, which records the *row_id* and *col_id* of the element);
- *num_changed*: number of elements changed during one iteration;
- *min*: the minimum # of reviewers that one team can have;
- *max*: the maximum # of reviewers that one team can have;
- *num_max*: the number of teams witch can have the maximum number of reviewers.

In this algorithm, let’s set $num\_max = r \times n \bmod t$, $min = r \times n/t$, and $max = min$ if num_max is equal to zero, otherwise, $max = r \times n/t + 1$.

Here is the ERee algorithm, which assign student *i* to do a review.

```
initialize matrix, team_review[n][t];
access the file storing the mapping information and put preexisting “1”s and “0”s
    into the matrix team_review[][];
calculate values for row_zeros[], row_ones[], col_zeros[], col_ones[] based on
    team_review[][];
if (row_enough_ones(i) = = 1)
        return; //This student has enough reviews.
back up this row (team_review[i][*]);
for each j such that team_review[i][j] = = “–1” {
        back up this column team_review[*][j];
        back up row_zeros[], row_ones[], col_zeros[], col_ones[];
        num_changed = 0;
```

```
        team_review[i][j] = 1;
        row_ones[i]++;
        col_ones[j]++;
        if (col_enough_ones(j) = = 1)
                check_col(j); // Would it be possible for i to review team j?

        if check_col(j) returned successful && row_enough_ones(i) = = 1 {
                // Back up the cells which may need to be restored later if this
                        choice is invalid
                for each remaining "–1" in this row (say its column id is j'){
                        team_review[i][j'] = 0;
                        row_zeros[i]++;
                        col_zeros(j')++;
                        //If we do this assignment, will it force some students to do
                        //1too many reviews?
                        if (col_enough_zeros(j') = =1 ) {
                                if check_zeros_col(j') is "unsuccessful"
                                        break; // and try another value for j
                        }
                }
        }

        if any of the above checks (check_col(j) or check_zeros_col(j')) was
                "unsuccessful" {
                restore the backed-up team and backed-up column;
                according to the row_id and col_id in the cell_changed[], set these
                     elements to "–1";
                restore row_zeros[], row_ones[], col_zeros[], col_ones[];
                team_review[i][j] = 0;
                row_zeros[i]++;
                col_zeros[j]++;
        else break; // we've found a reviewee j.
        }
}
return;
```

Here are 8 subroutines used by the above procedure:

1). *check_row*(**int** *i*): there is enough "1"s in i^{th} row, so set remaining "-1"s in this row to "0"s

2). *check_zeros_row*(**int** *i*): there are enough "0"s in i^{th} row, so set remaining "-1"s in this row to "1"s

3). *check_col*(**int** *j*)): there is enough "1"s in j^{th} column, so set remaining "-1"s in this column to "0"s

4). *check_zeros_col*(**int** *j*): there are enough "0"s in j^{th} column, so set remaining "-1"s in this column to "1"s

5). *row_enough_ones*(**int** *i*): check whether there is **enough** "1"s in i^{th} row

6). *row_enough_zeros*(**int** *i*): check whether there are **enough** "0"s in i^{th} row

7). *col_enough_ones*(**int** *j*): check whether there is **enough** "1"s in j^{th} column

8). *col_enough_zeros*(**int** *j*): check whether there are **enough** "0"s in j^{th} column

The last four subroutines are for the flexibility of this team review mapping strategy. In the following part of this chapter, we can see that the only difference between EReе Algorithm and the ERer algorithm is in the difference of those last four subroutines. After these four functions are introduced, all of the other functions above, including the main function, can be used in ERer algorithm. It makes the functions very flexible to implement.

```
check_row(int i) {
// Precondition: This reviewer has enough teams to review. So, set
// remaining "–1"s to "0"s
        for each remaining "–1" in this row (say its column ID is j) {
                cells_changed[num_changed].row_id = i;
                cells_changed[num_changed].col_id = j;
                num_changed ++;
                team_review[i][j] = 0;
                row_zeros[i]++;
                col_zeros[j]++;
                if (col_enough_zeros(j) = = –1)
                        return "unsuccessful";
                else if (col_enough_zeros(j) = = 1)
                        // Will this cause a reviewer to do too many reviews?
                        check_zero_col(j);
                        if (check_zero_col(j) returns "unsuccessful")
```

```
                    return "unsuccessful";
        }
        return "successful"
}

check_zeros_row(int i) {
// Precondition: This reviewer has (n - r) invalid mapping assignments. So, set
// remaining "–1"s to "1"s
        for each remaining "–1" in this row(say column ID is j) {
                cells_changed[num_changed].row_id = i;
                cells_changed[num_changed].col_id = j;
                num_changed ++;
                team_review[i][j] = 1;
                row_ones[i]++;
                col_ones[j]++;
                if (col_enough_ones(j) = = –1)
                        return "unsuccessful";
                else if (col_enough_ones(j) = = 1)
                        // Will this cause a reviewer to do too few reviews?
                        check_col(j);
                        if check_ col(j) return "unsuccessful"
                                return "unsuccessful";
        }
        return "successful"
}

check_col(int j) {
// Precondition: All reviewers for this submission have been mapped. So, set
// remaining "–1"s to "0"s
        for each remaining "–1" in this column (say row ID is i) {
                cells_changed[num_changed].row_id = i;
                cells_changed[num_changed].col_id = j;
                num_changed++;
                team_review[i][j] = 0;
                row_zeros[i]++;
                col_zeros[j]++;
                if (row_enough_zeros(i) = = –1)
                        return "unsuccessful";
                else if (row_enough_zeros(i) = = 1)
                        // Will this cause a team to have too many reviewers?
                        check_zeros_row(i);
                        if check_zeros_row(i) return "unsuccessful"
                                return "unsuccessful";
        }
        return "successful";
}
```

```
check_zeros_col(int j) {
// Precondition: This reviewee already has enough impossible choices. So, set
// remaining "–1"s to "1"s
        for each remaining "–1" in this column (say row ID is i) {
                cells_changed[num_changed].row_id = i;
                cells_changed[num_changed].col_id = j;
                num_changed++;
                team_review[i][j] = 1;
                row_ones[i]++;
                col_ones[j]++;
                if (row_enough_ones(i) = = –1)
                        return "unsuccessful";
                else if (row_enough_ones(i) = = 1)
                        // Will this cause a team to have too few reviewers?
                        check_row(i);
                        if (check_row(i) return "unsuccessful")
                                return "unsuccessful";
        }
        return "successful";
}

row_enough_ones(int i) {
// This function is used to check whether student i has enough valid reviewees or
// not
// return "1" if # of "1"s in row i is enough
// return "–1" if # of "1"s in row i  is more than enough
// return "0" otherwise
        if(row_ones[i] = = r)
                return 1;
        else if(row_ones[i] > r)
                return –1;
        else
                return 0;
}

row_enough_zeros(int i) {
// This function is used to check whether student i has enough invalid reviewees
// or not
// return "1" if # of "0"s in row i  is enough
// return "–1" if # of "0"s in row i  is more than enough
// return "0" otherwise
        if(row_zeros[i] = = t - r)
                return 1;
        else if(row_zeros[i] > t - r)
                return –1;
```

```
        else
                return 0;
}

col_enough_ones(int j) {
// This function is used to check whether team j has enough valid reviewers or not
// return "1" if # of "1"s in column j  is enough
// return "–1" if # of "1"s in column j  is more than enough
// return "0" otherwise
        count # of column j' with col_ones[j'] = = max; (Say the number is count)
        if (count = = num_max) {
        // there are enough teams have been reviewed max times
                if (col_ones[j] = = min)
                        return 1;
                else if (col_ones[j] > min)
                        return –1;
                else return 0;
        }
        if (count < num_max) {
        // there are not enough teams have been reviewed max times
                if (col_ones[j] = = max)
                        return 1;
                else if (col_ones[j] > max)
                        return –1;
                else return 0;
        return –1;
        }
}

col_enough_zeros(int j) {
// This function is used to check whether team j has enough invalid reviewers or
// not
// return "1" if # of "0"s in column j  is enough
// return "–1" if # of "0"s in column j  is more than enough
// return "0" otherwise
        count # of column j' with col_zeros[j'] = = n - min; (Say the number is
                count)
        if (count = = t – num_max) {
        // there are enough teams have (n – min) number of students that can't
        // review this submission
                if (col_zeros[j] = = n-max)
                        return 1;
                else if (col_zeros[j] > n- max)
                        return –1;
                else return 0;
        }
```

```
        if (count < t – num_max) {
        // there are not enough teams have (n – min) number of students that can't
        // review this submission
                if (col_zeros[j] < n-min)
                        return 1;
                else if (col_zeros[j] > n- min)
                        return –1;
                else return 0;
        }
        return –1;
}
```

2.3.ERer Algorithm

2.3.1. Outline of ERer Algorithm

Now after we have described the EReе Algorithm, let's take a look at the ERer Algorithm. ERer stands for equal number of reviewers, which means that each team needs to have the same number of reviewers.

The basic idea the of ERer Algorithm is similar to the EReе Algorithm. The matrix we used in EReе Algorithm, *team_review*[1:*n*][1:*t*], is also used in the ERer Algorithm. Let's assume that in the class we mentioned above, the number of students is still *n*, and the number of teams is still *t*. What has changed is that each team needs *r* reviewers.

Definition 1, Definition 2, and Definition 4 are valid for the ERer Algorithm.

Definition 5. A mapping assignment of a reviewer to a team is *valid* in the ERer Algorithm if each student has the same number of reviewees, as nearly as possible.

To meet the requirement of Definition 5, three numbers are needed, the maximum number (*max*) of reviewees that one student can have, the minimum number (*min*) of reviewees, and the number of students (*num_max*) who can have *max* teams to review. To make sure each team is reviewed by the same number of individuals, as nearly as possible, we assume *max* is equal to *min* if *num_max* is equal to 0; otherwise, max is 1 larger than min.

The basic logic of this algorithm is to prevent the number of "1"s becoming larger than *r* in any column, and the number of "1"s in any row becoming greater than *max*. Otherwise, it would be an invalid mapping assignment. Similarly, the number of "0"s cannot be greater than ($n - r$) in any column, and the number of "0"s in any column can't be greater than ($t - min$).

Definition 6. In the ERer Algorithm, one row/column has **enough** "1"s if the number of "1"s in this row/column has reached the limit. Each row's limit is *max* if the number of students who have *max* reviewees is smaller than *num_max*; otherwise, this number is *min*, and each column's limit is *r*.

Definition 7. In the ERer Algorithm, we say that one row has **enough** "0"s if the number of "0"s in this row reaches ($t - max$), or ($t - min$) when max reviewers is not acceptable for this student (there are already *num_max* students that have max reviewees), and one column has **enough** "0"s if the number of zeros in this column reaches ($n - r$).

2.3.2. Outline of ERer Algorithm

All of the variables and functions in the ERer Algorithm are the same as those in ERer Algorithm, except four functions and the initial values of three important variables.

Here are those three important variables.

- *min*: the minimum # of teams that one student can review;
- *max*: the maximum # of teams that one student can review;
- *num_max*: the number of students who can review the maximum number of teams.

In this algorithm, let's set $num\_max = r \times t \bmod n$, $min = r \times t/n$, and $max = min$ if *num_max* is equal to zero; otherwise, $max = r \times t/n + 1$.

As we said above, there are four functions that make the ERer algorithm different from the ERee algorithm. These four functions are to check whether "1"s or "0"s in one row/column are enough. We know, the difference between ERee Algorithm and ERer Algorithm is that ERee Algorithm is for a situation where each student has the same number of reviewees, but the ERer Algorithm is for the situation where each team has the same number of reviewers. That means, in ERee Algorithm, each row should have the same number of "1s"/"0"s with other rows (*r*), but each column might have a different number of "1"s/"0"s with other column (*min* or *max*). In ERer Algorithm, each row might have a different number of "1s"/"0"s than other rows (*min* or *max* "1"s in each row), but

each column should have the same number of “1”s/“0”s as other columns (*r* “1”s in each column). Therefore, there are different standards to determine whether one row/column has enough “0”s or “1”s in the ERee Algorithm and ERer Algorithm, and that’s why those four functions are different in the ERee Algorithm and the ERer Algorithm.

```
row_enough_ones(int i) {
// This function is used to check whether student i has enough valid reviewees or
// not
// return “1” if # of “1”s in row i  is enough
// return “–1” if # of “1”s in row i  is more than enough
// return “0” otherwise
        count # of row i′ with row_ones[i′] = = max; (Say the number is count)
        if (count = = num_max) {
        // there are enough students have reviewed max teams
                if (row_ones[i] = = min)
                        return 1;
                else if (row_ones[i] > min)
                        return –1;
                else return 0;
        }
        if (count < num_max) {
        // there are not enough students have reviewed max teams
                if (row_ones[i] = = max)
                        return 1;
                else if (row_ones[i] > max)
                        return –1;
                else return 0;
        }
        return –1;
}

row_enough_zeros(int i){
// This function is used to check whether student i has enough invalid reviewees
// or not
// return “1” if # of “0”s in row i  is enough
// return “–1” if # of “0”s in row i  is more than enough
// return “0” otherwise
        count # of row i’ with row_zeros[i’] = = t - min; (Say the number is count)
        if (count = = n – num_max) {
```

```
        // there are enough students have (n – min) invalid assignment
                if (row_zeros[i] = = t-max)
                        return 1;
                else if (row_zeros[i] > t- max)
                        return –1;
                else return 0;
        }
        if (count < n – num_max) {
        // there are not enough students have (n – min) invalid assignment

                if (row_zeros[i] = = t-min)
                        return 1;
                else if (row_zeros[i] > t- min)
                        return –1;
                else return 0;
        }
        return –1;
 }

col_enough_ones(int j) {
// This function is used to check whether team j has enough valid reviewers
// return "1" if # of "1"s in column j  is enough
// return "–1" if # of "1"s in column j  is more than enough
// return "0" otherwise
        if (col_ones[j] = = r)
                return 1;
        else if (col_ones[j] > r)
                return –1;
        else return 0;
}

col_enough_zeros(int j) {
// This function is used to check whether team j has enough valid reviewers or not
// return "1" if # of "0"s in column j  is enough
// return "–1" if # of "0"s in column j  is more than enough
// return "0" otherwise
        if (col_zeros[j] = = n – r)
                return 1;
        else if (col_zeros[j] > n – r)
                return –1;
        else return 0;
}
```

CHAPTER 3: IMPLEMENTATION AND PERFORMANCE ANALYSIS OF DYNAMIC TEAM PEER-REVIEW MAPPING ALGORITHMS

3.1. Implementation

ERee Algorithm and ERer Algorithm have been implemented in Java, and tested in the environment of jdk1.3. The interface of these two algorithms is listed below. The interface is implemented by both of ERee algorithm and ERer algorithm. We can see the important methods needed to implement those two algorithms, and have an impression about how ERee algorithm and ERer Algorithm work. Most of the methods in this interface have been introduced in Chapter 2.

```
public interface TeamReview{
        // load information of students
        void read_data();

        // initialize the matrix team_review[][]
        void initialize();

        // get the team number of student "stu"
        int team_num(int stu);

        // get a reviewee for student "stu"
        void map_one_stu(int stu);

        // set all of the remaining "-1"s in the row "row" to "0"s
        boolean check_row(int row);

        // set all of the remaining "-1"s in the row "row" to "1"s
        boolean check_zeros_row(int row);

        // set all of the remaining "-1"s in the column "col" to "0"s
        boolean check_col(int col);

        // set all of the remaining "-1"s in the column "col" to "1"s
        boolean check_zeros_col(int col);

        // check whether there are enough “1”s in row “row”
```

```
        int row_enough_ones(int row);

        // check whether there are enough “0”s in row “row”
        int row_enough_zeros(int row);

        // check whether there are enough “1”s in column “col”
        int col_enough_ones(int col);

        // check whether there are enough “0”s in column “col”
        int col_enough_zeros(int col);
}
```

3.2. Performance Analysis

In this part, let’s pick the ERee Algorithm, analyze its performance, and see how parameters such as number of students, number of teams and number of reviews number, affect the execution time. Here are the parameters used in the figures below:

n: number of students in the class;

t: number of teams in the class;

r: number of teams each student need to review;

time: execution time with n students, t teams, and r reviewees per student .

According to the analysis of the algorithm, we find the relationship between the execution time and those three factors (*n*, *t*, *r*) is O (n^2t^3r) in the worst case. The figures of the changes of execution time are showed below.

3.2.1. Effect of Number of Students (O (n^2))

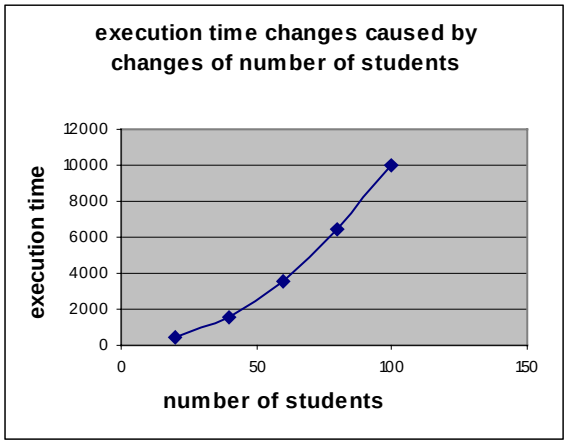


Figure 1: Execution time vs. number of students

Assume that the number of teams and the number of reviews per student keep unchanged, the figure of the relationship of execution time and number of students in the worst case is showed above.

3.2.2. Effect of Number of Teams (O (t^3))

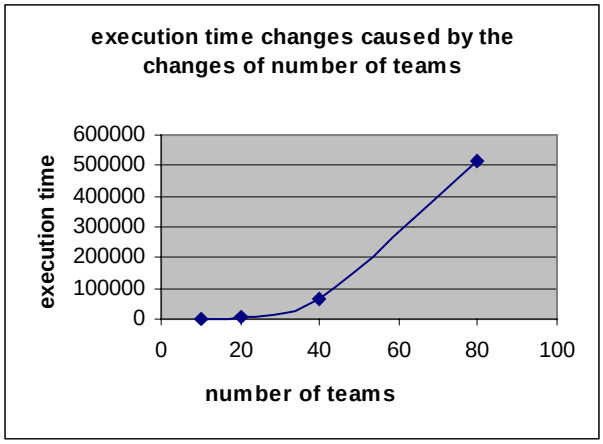


Figure 2: Execution time vs. number of teams

Assume that the number of students and the number of reviews per student keep unchanged, the figure of the relationship of execution time and number of teams in the worst case is showed above.

3.2.3. Effect of Number of Reviews

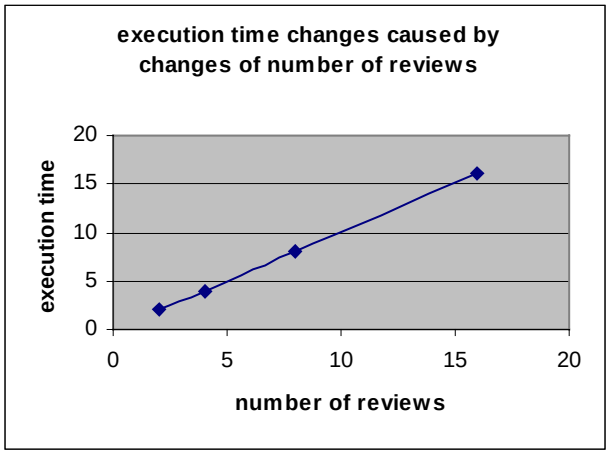


Figure 3: Execution time vs. number of reviews

Assume that the number of students and the number of teams keep unchanged, the figure of the relationship of execution time and the number of reviews per student in the worst case is showed above.

CHAPTER 4: SUMMARY AND CONCLUSION

In this thesis, we have studied strategies for dynamic team peer review mapping. Individual peer review can be considered as a kind of team peer review with one and only one member in each team, so this strategy can be used both for team peer review and individual peer review. A random mapping arranged statically in advance is frustrated by students who drop the course during the assignment period, or who do not submit their assignments; that's why we developed this strategy. This dynamic strategy makes a mapping assignment when a student asks to review, and ensures each team has the same number of reviewers as nearly as possible when each student is asked to do the same number of reviews, or each student can review the same number of teams as nearly as possible when each team is required to be reviewed the same number of times. By using this dynamic peer review mapping strategy, no students are assigned to review their own team.

When multi-member teams are being reviewed, the number of reviews a team gets is *not* equal to the number of reviews done by each reviewer. In fact, depending on the size of the teams and the number of reviewers, it may not be possible both to have each reviewer review the same number of teams *and* have each team reviewed by the same number of reviewers. Thus, with team review, two algorithms for two situations are developed. When each reviewer is required to review the *same number of teams*, the ERee (equal number of reviewees) algorithm is used. On the other hand, when each team needs to get the *same number of reviewers*, it's time for the ERer (equal number of reviewers) algorithm.

The ERee algorithm is introduced first. As we said on the above, ERee stands for equal number of reviewees, which means that each reviewer will review the same number of teams (let's say the number is *r*). In the ERee Algorithm each team is required to be reviewed by the same number of individuals, as nearly as possible. A matrix, *team_review*[1:*n*][1:*t*], is used in ERee Algorithm (assume there are *n* students and *t* teams in this class). In this matrix, each row is for a particular reviewer, and each column is for a particular team (submission). The status of one mapping assignment is shown by the corresponding value in this matrix. A valid assignment is represented by "1". An invalid assignment is represented by "0", and "–1" means this assignment is not be mapped yet. The basic logic of this algorithm is to prevent—

- the number of "1"s from becoming larger than *r* in any row, and
- the number of "1"s in any column becoming greater than *max* (the maximum number of reviewers one team can have).

When one student asks to do a review, we pick one element in his/her corresponding row with the value of "-1", and set it to "1" if the corresponding assignment is submitted. Then we check whether this assignment is valid or not. If it's valid, this student gets a team to review, or we select another possible element. Following the outline of ERee algorithm, an example of how this algorithm works is introduced. Then the functions of ERee algorithms are listed.

The ERer algorithm is described after ERee algorithms. ERer stands for equal number of reviewers, which means that each team needs to have the same number of reviewers. In the ERer Algorithms, each student is required review the same number of teams, as nearly as possible. The basic idea the of ERer Algorithms is similar to the ERee Algorithms.

The matrix we used in ERee Algorithms, *team_review*[1:*n*][1:*t*], is also used in the ERer Algorithms (Let's assume that in the class we mentioned above, the number of students is still *n*, and the number of teams is still *t*. What has changed is that each team needs *r* reviewers). What make the ERer algorithms different from the ERee algorithms are functions to check whether "1"s or "0"s in one row/column are enough. We know the difference between ERee algorithms and ERer is that the ERee algorithm is for situation where each student has the same number of reviewees, but the ERer algorithm is for the situation where each team has the same number of reviewers. That means, in ERee algorithm, each row should have the same number of "1s"/"0"s with other rows (*r*), but each column might have different number of "1"s/"0"s with other column (*min* or *max*). In the ERer algorithm, each row might have different number of "1s"/"0"s with other rows (*min* or *max*), but each column should have the same number of "1"s/""0s with other column (*r*). Therefore, there are different standards to determine whether one row/column has enough "0"s or "1"s in ERee algorithm and the ERer algorithm, and that's why those four functions are different in the two algorithms. The existence of those four functions makes it very easy to implement these two algorithms. We list those four functions after describing ERer Algorithm.

Finally, the interface of these two algorithms is listed. After analyzing the performance and the effects of some important parameters, such as number of students (*n*), number of teams (*t*), and number of reviews (*r*), we see that in the worst case, the relationship of execution time and those three factors is O (n^2t^3r). The figures about the execution time and those three factors are showed.

Future work

The strategy we presented in this thesis can produce a valid mapping assignment for reviewers dynamically. However, in certain circumstances, this strategy would produce invalid assignments.

If one student drops this course after one mapping is assigned, it may produce one invalid assignment. Assume that there are four students and four teams with one student in each team, and our algorithm would have produced the mapping (S_1 reviews S_2, S_2 reviews S_1, S_3 reviews S_4, and S_4 reviews S_3). If the first two mapping assignments ($S_1 \rightarrow S_2$ and $S_2 \rightarrow S_1$) have already been made, and S_4 drops the course, we have no alternative but to assign S_3 to review himself.

There is another situation that might produce an invalid assignment. For example, in one class there are seven students (S_1, S_2, S_3, S_4, S_5, S_6, S_7) who work in three teams (T_1, T_2, T_3), and each student needs to review two teams. Let's say S_1, S_2, S_3, S_4 work on T_1, S_5, S_6 work on T_2, and S_7 works on T_3. In this case, $n = 7$, $t = 3$, and $r = 2$, so $min = 7 \times 2/3 = 4$, and $max = 5$. According the requirement of the ERee algorithm, the number of reviewers per team should be no less than *max* (5) and no larger than *min* (4). We can easily see that to make each student review two teams, everyone should review two teams beside their own team; that is each team should be reviewed by all of the students who are not on the team. Thus, T_1 has three reviewers (less than *min*), T_2 has five reviewers, and T_3 has six reviewers (lager than *max*), which makes several assignments invalid.

These two cases happen in the situation where there are few students in one class and the numbers of students in teams vary much, so the probability of them occurring in

practice is very small. Further work is necessary to see how to minimize the probability that an invalid mapping will result.

References

[1] [DB 97] Downing, T., and Brown, I., "Learning by cooperation publishing on the World Wide Web," *Active Learning* 7 (1997), pp. 14–16.

[2] [GC 02] Gehringer, Edward F., Yun Cui, "An Effective Strategy for Dynamic Mapping of Peer Reviewers," 2002 ASEE Annual Conference and Exposition, American Society for Engineering Education, Albuquerque.

[3] [Gehr 01] Gehringer, Edward F., "Assignment and quality control of peer reviewers," 2001 ASEE Annual Conference and Exposition, American Society for Engineering Education, Albuquerque, June 26, 2001

[4] [KPD 95] Kerr, Peter M., Park, Kang H., and Domazlicky, Bruce R., " Peer grading of essays in a principles of microeconomics course," *Journal of Education for Business* 70:6, July 1995, pp. 357 ff.

[5] [SGG 01] Silbershatz, Abraham, Galvin, Peter, and Gagne, Greg, *Operating System Concepts*, 6th ed., John Wiley and Sons, 2001.

[6] [Sys 98] Styslinger, Mary E., "Some milk, a song, and a set of keys: Students respond to peer revision," *Teaching and Change* 5:2 (Winter 1998), pp. 116–138

[7] [Sys 99] Styslinger, Mary E., "Mars and Venus in my classroom: Men go to their caves and women talk during peer revision," *English Journal* 88:3 (Jan. 1999), pp. 50-55

[8] [Topp 98] Topping, Keith, "Peer assessment between students in colleges and universities," *Review of Educational Research* 68:3 (Fall 1998), pp. 249-276.