Automating Academic Advising Assignments

Andrew Banister, Salma Chiheb, John Daniels, Heather Gronewald, Jordan King, Paris Nelson

Departments of Computer Science, Economics, and Mathematics

Southwestern University

Georgetown, Texas 78626 USA

April 30, 2014

## 

**Abstract**

We provide a program that improves efficiency of the academic advising assignment process at Southwestern University, while maximizing compatibility between advisors and advisees. Part of the academic experience at Southwestern is working closely with an academic advisor (a faculty or professional staff member trained to offer guidance through degree requirements, career planning, and academic and extracurricular goals). Advisor assignments are currently made manually; existing procedures are inconsistent, unnecessarily time-consuming, and sub-optimal. We propose a linear optimization program that considers advising load constraints and maximizes compatibility between advisors and advisees based on quantified and prioritized factors (including potential transitional challenges and intended major). Because ideal assignments cannot be guaranteed for each incoming student, our program optimizes the aggregate incoming class compatibility score, then outputs metrics about individual ideal pairings as well as aggregate class compatibility.

**Executive Summary**

**Overview**

We present a program that assigns incoming students to available faculty/staff advisers at Southwestern University, maximizing compatibility between advisees and advisors. Advisor assignments are currently made manually; this method is inconsistent, unnecessarily time-consuming, and sub-optimal. Our program automates this process based on quantified and prioritized factors (including intended major, potential transitional challenges, and availability of advisors) and optimizes compatibility.

**Solution**

Our objective is to optimize the assignment process; we propose a program that maximizes the number of students that are in ideal and secondary advising placements within the constraints of feasibility. Our project is also beneficial in that it automates the pairing process (increasing efficiency), clarifies existing variables used by the CAS (articulates and documents the process), and produces a system that outputs quantifiable feedback on each incoming class’s advising match-ups (allows for assessment).

**Model**

Our basic model is a transportation problem that uses a flowchart (See Appendix in Final Report), provided in the appendices, to represent the ideal distribution routes of incoming advisees (ideal meaning that there are enough qualified faculty for all advisees). The flowchart should adequately sort a majority of students to the appropriate groups for use in the transportation problem.

We plan to quantify the performance of the distribution model via the number of students who are placed with an advisor with which they are compatible. We define compatibility through discrete levels (advisor in your major>advisor in your division>advisor in your department>advisor unrelated to area of interest + appropriate level of training), with the appropriate level of training weighted higher than any of the other criteria. Given this quantification, our model attempts to maximize the number of strong or perfect matches.

**Transportation Problem**

A basic transportation problem consists of sources with supply, destinations with demand, and distinct positive values associated with a particular source supplying a particular destination. The objective is to satisfy all demands on the existing supply in a way that maximizes total value.

Value is calculated by adding points for meeting specific criteria to assess the strength of a match. To enforce the overflow strategy discussed previously, the points are allotted as below:

Worth 1 point:

•Correct Division/School

•Correct Department

•Correct Major

Worth 5 points:

•Advisor at or above beneficial level of TC preparedness

These values are then squared in order to increase the discrepancy between stronger and weaker matches. On a problem of this size we believe this is an important consideration.

**Team**

Andrew Banister, Mathematics major, Southwestern University

Salma Chiheb, Business major, Southwestern University

John Daniels, Computer Science major, Southwestern University

Heather Gronewald, Mathematics/English majors, Southwestern University

Jordan King, Computer Science major, Southwestern University

Paris Nelson, Computer Science major, Southwestern University

**Results**

We have developed software that, given aggregated data on an incoming class of students, can automatically assign these students to appropriate advisors in a way that maximizes total compatibility. In order to automate the pairing process we have worked with the CAS to quantify the existing questionnaire. With additional work on defining TC, our software is capable of interacting with the new online format of the questionnaire. This software will greatly reduce the amount of manual work that has been used in the past as well as decrease the amount of time needed to complete the assignment process.

**Conclusions**

The current implementation of the program works as we have defined. It is able to account for different sets of students and advisors along with their associated majors, capacities, and levels of training. To fully integrate the program into a usable system for the CAS, further work must be done to expand the program to translate TC into a numerical value and to accommodate for advisors with different capacities. The parameter values can also be further optimized by testing past, future, or fictitious student data. Implementation of our program will ease stress on CAS, while providing better matches for advisors and advisees.

## 1 Introduction

**1.1 Motivation**

Part of the academic experience at Southwestern is working closely with an academic advisor, a faculty or professional staff member trained to offer guidance through degree requirements, career planning, and academic and extracurricular goals. Academic advisors significantly impact the educational experience of Southwestern students. Thus, it is critical that the experience and training of advisors be compatible with respective advisee goals and needs. However, individual faculty and staff have limits on the number of advisees they can handle, so ideal assignments cannot be made for each incoming student. The distribution of advisees should consider each incoming student’s needs and then optimize the collective class compatibility with advisors.

Currently, the Center for Academic Success (CAS) makes advising assignments manually: students’ goals and potentials for transitional challenges are primarily evaluated qualitatively, leading to pairings with appropriate advisors. An advisor who has reached his or her capacity for students is removed from the pool of potential advisors. A student ideally matching to a removed advisor is considered overflow, and cascades from an ideal individual advisor to a major, then to a department, and then to a division (e.g. a student ideally matched to Dr. Marr would overflow to the Mathematics major, then to the Mathematics and Computer Science department, and then to the Natural Sciences division). Further surplus is distributed arbitrarily among any faculty/staff that are not at advising capacity until all students have paired with advisors.

Although an impressive amount of personal attention is given to each incoming student’s advising questionnaire and subsequent advisor-pairing, the current method is inconsistent and unnecessarily time-consuming (roughly six staff work on pairings over ten work days). People making advising assignments may carry unconscious bias into pairing decisions, and six staff considering roughly 400 incoming students are not able to collect and retain all incoming student information in order to truly optimize advising compatibility. Manual evaluation and distribution of advising assignments require significant training for staff. Most importantly, there is no real method for evaluating advising pairings; however, roughly half of Southwestern students change advisors.

Refining the academic advising pairing process to improve both efficiency and outcome is advantageous to the CAS. Future Southwestern students will directly benefit from optimal advisor assignments. Faculty and staff serving as advisors also benefit from working with students they are most adequately prepared to support.

**1.2 Objective**

Our primary objective is to optimize compatibility between students and advisors. Tertiary objectives are to automate the pairing process (increasing efficiency), clarify existing variables used by the CAS (articulate and document the process), and provide quantifiable feedback on each incoming class’s advising match-ups (allow for long-term assessment).

A computer program enforces only written constraints, so by automating the advising pairing process, we intend to eliminate unconscious bias. Quantifying and documenting the methods for distributing advisees ensures that procedures are both visible and accountable to the campus community. Further, in the event of key (or high) staff turnover, an automated system remains operable with little transitional training difficulties.

Additionally, we hope to decrease advisor changes. Students request changes for three primary reasons: personality conflicts with advisors, changes in areas of interest (declared majors), and inappropriate or insufficient addressing of students’ interests/needs. We cannot influence the first two reasons, which are unpredictable and unavoidable, but intend to decrease advisor-change requests that are symptomatic of poor assignments.

**1.3 Overview**

In order to model this problem, we worked with Kim Murphy to create a flowchart simulating the ideal distribution of incoming students; this both clarified priorities for advising assignments and allowed us to group incoming students with identical advising needs. We then assigned values to our existing criterion for advising compatibility (e.g. advisor in advisee’s intended major) and used an existing linear programming method (a transportation problem) to maximize aggregate compatibility between advisors and advisees.

## 2 Model Formulation

## 2.1 Defining the Problem

In order to optimize compatibility between students and advisors, we have to quantify the variables that makes students and advisors compatible. Currently students write responses to an advising questionnaire (see Advising Questionnaire in appendix attached); though some variables (like intended major) are easily programmable, others are more qualitative in nature, and are difficult to program efficiently and consistently. However, defining quantitative measurements is necessary for the automation process. The CAS is in the process of creating a Google Form for their questionnaire, and we have provided them with feedback to facilitate a transition to collecting quantifiable data whenever possible. We have also received information from the CAS on ways to quantify existing variables.

**2.1.1 Transitional Challenges**

According to the CAS, some characteristics of incoming students positively correlate with transitional challenges. Transitional challenges are indicators that a student may have a more challenging time adapting to the academic and social environment of Southwestern; the CAS attempts to match students with varying levels of transitional challenges to advisors with appropriate levels of experience and training. Examples of transitional challenges include:

* Rural HS (quantify by size threshold)
* Graduation in the lower 25% of high school class
* First generation college student
* Academic concerns (self-reported by students)
* Late deposit

The CAS provided us with a list of faculty that have the training and experience to work with students with moderate transitional challenges. We set thresholds that allow us to classify students into one of three categories: high transitional challenges ideally match to CAS advisors (5+ TCs), some transitional challenges ideally match to specified faculty with appropriate training (2-4 TCs), and no significant transitional challenges ideally match to any faculty (0-1 TCs). These three transitional challenge levels are included in our flowchart model.

**2.1.2 Majors, Departments, and Divisions**

In addition to transitional challenges, the CAS considers students declared major or declared area of interest when making assignments. Declared major and area of interest are considered identical by the CAS. Ideally a student is assigned to an appropriately trained faculty member in their declared major, space permitting. Overflow is assessed from major to department then from department to division (or school in the case of the Sarofim School of Arts). This overflow scheme aims to increase the likelihood that advisors are familiar with degree plan or the subject matter that a student aims to pursue.

**2.1.3 Other Considerations**

Some students are handled in special ways due to the circumstances of their admittance to Southwestern. Transfer students are automatically sent to a transfer advisor within the CAS who has more experience with credit transferral. International students are assigned to a faculty member in the language department who may be able to have more productive communication. Athletes who have not indicated an area of interest are assigned to an off-season coach who may have more experience working with academic schedules. These special cases are handled in the context of the flowchart and are sorted out of the incoming pool of students before the general assignment process is completed.

**2.2 Assumptions**

We define a strong match as a student matched with an advisor in the student’s indicated major, along with the appropriate training based on the students’ TC level. We assume an advisor matched with an advisee in the same area of interest or declared major is a better match due to a greater understanding of degree plan, including appropriate prerequisites and suggested courses. For the same reasons we assume the next best match is same department, then the same division.

Students with similar interests and levels of transitional challenges will be equally satisfied with the same advisor. Likewise, advisors in the same department with the same amount of training are assumed to be equal. Advisors are labeled to either be prepared to handle TC or not. We assume that staff at the Center for Academic Success are the most equipped to handle TC, and that other faculty and staff can be categorized in one of two ways: capable of handling moderate levels of transitional challenges (through training and experience) and incapable of effectively handling moderate transitional challenges. We accept Kim Murphy’s division of faculty and staff into these two categories.

In our current model, TC are considered to have equal weight when determining a student’s level. No specific characteristic or combination of characteristics is considered more challenging than any other. A corollary assumption is that all faculty are equally well prepared to offer guidance on any TC, and are not trained for any TC in particular.

We assume the TC thresholds (low, moderate, and high) described above based on limited verbal feedback from the CAS. While this is a large assumption, it can be easily refined within the program.

Because advising match-ups are based largely on a student questionnaire, we must assume that students can adequately assess and express variables indicating transitional challenges. To allow for inconsistent self-reporting, specific variables can be checked against student responses. For example, a student that expresses interest in improving writing skills, time-management, and study habits, but graduated in the top five percent of his or her high-school class should not be classified as having significant transitional challenges. Our existing model provides an adequate baseline for evaluating transitional challenges; we can easily add constraints as needed.

The current model assumes each advisor has a uniform fixed capacity of 5 students. However, many advisors begin the school varying advising loads and there are students that graduate and professors that leave on sabbatical. Our program accommodates varying advisor capacities, but they need to be input before implementation. We also assume that there are sufficient faculty to support the students.

## 2.3 Model

Our basic model is a transportation problem that uses a flowchart, provided in the appendix, to represent the ideal distribution routes of incoming advisees (based on feedback from the CAS). The flowchart sorts students into appropriate demand groups for our transportation problem.

Students are grouped by major of interest and level of transitional challenges; similarly, advisors are grouped by the major in which they advise and their preparedness for handling transitional challenges. These groupings are performed as part of our assumption that students within the same major with the same number of transitional challenges value advisors in the same way. This grouping process also has the benefit of greatly reducing the number of variables and constraints needed for the linear program, which speeds up the process of optimizing the assignments.

We assess the performance of the distribution model via the number of students who are placed with an advisor with which they are compatible and their level of compatibility. We define compatibility through a score where points are accumulated for an advisement meeting certain criteria. The criteria used for this score and their associated point values are as follows:

Worth 1 point:

•Correct Division/School

•Correct Department

•Correct Major

Worth 5 points:

•Advisor at or above beneficial level of TC preparedness

Note that being in the correct department and division are inherent to being in the correct major. Therefore, being in the correct major is essentially worth 3 points (one for each of major, department, and division). Also being assigned to an advisor with the correct training is given more points than the other criteria as a result of the CAS’s belief that this criterion is more important than being assigned to an advisor in the correct major. Therefore our model prioritizes students receiving the correct level of assistance for transitional challenges. Rather than maximizing the quantity of ideal matches, our model maximizes the sum of point values for all assigned students. This lends itself to linear programming in the form of a transportation problem, which can maximize quantified student compatibility values.

### 2.3.1 Transportation Problem

A basic transportation problem consists of sources with supply, destinations with demand, and distinct positive values associated with a particular source supplying a particular destination. The objective is to satisfy all demands on the existing supply in a way that maximizes total value. The assignment process can be modeled this way by organizing students with similar characteristics using criteria outlined in the flowchart (such as indicated major/interest and level of transitional challenges) into individual demand groups. Faculty are grouped by major and level of training; these groups become the sources and their supply is equal to the capacity of students they could take on as advisees. These roles are appropriate as students must be assigned advisors (demand must be met) but advisors do not have to advise the maximum possible number of students (slack in supply).Values are determined by the compatibility of a match assessed by the score defined above. The objective is to optimize the overall strength of assignments.

**3 Solving the Model**

We formulated a model of the assignment process using a transportation problem. We leveraged GLPK (GNU Linear Programming Kit), software for solving large scale linear programs and mixed integer programs. The GLPK uses a program called GLPSOL which is invoked via command line to solve LP's and IP's. This software was appropriate for our uses primarily because our LP contains a number of constraints equal to the number of advisor groups (~75) plus the number of student groupings (~125) and also non-negativity and integer constraints. This many constraints make the problem infeasible for solution via Microsoft Excel's LP Solver. In addition, GLPK can handle integer programs, executes quickly, and can be called from command line where file names can be provided for the LP's solution and sensitivity analysis to be written. The significance of this last benefit is that the entire solution process can be managed from the command line: writer program (writer/reader will be discussed in detail below) can be called to write the LP, GLPSOL can be used to solve the LP, and the reader program can be called to interpret and present the results. This process is made feasible by our approach to solving our model in a generalized, automated, and easily maintainable manner.

Input to our program is provided in the form of several text files. The first is a file containing a list of line-separated majors available at Southwestern and their associated department and division. The second is a file containing comma-separated advisor data including (on each line) the advisor's name, their associated major (multiple majors are allowed), and whether or not they are trained to handle transitional challenges (encoded numerically). The final input file needed is a comma-separated list of student data including (again, per line) the student's name, their intended major (undecided is an option), whether they are a transfer student, an international student, or neither, and how many transitional challenges they reported. Both of the comma-separated files can be created easily from an Excel spreadsheet, which is the format we expect the data to be held in.

This implementation has the advantage of being adaptable; nothing is assumed about faculty or student data except the format in which it is provided. Since the available faculty and their respective training levels will naturally change from year to year, as will the incoming student data, it is beneficial for the program to be able to take in new data every year it is run. Getting the data into the required format is also convenient as the current plan for the 2014-15 incoming class is to survey the students using a Google form which will aggregate all of the students' data into a single Excel spreadsheet. From there it is simple to convert the data into a text file of comma-separated values that can be fed into our program.

Given the input has been provided, our program uses the data to create a linear program. Students are grouped by their major of interest and the level of transitional challenges they claim. Advisors are grouped by major(s) in which they advise. The program then creates a modified transportation problem table where the student groups represent demand (destination) and the advisors groups are the supply (source). In the createVariablesAndScores() method, the score for each assignment is calculated based on the rules discussed above in the 'Transportation Problem' section. In our program, this table is represented by 2 two-dimensional arrays: one holds the compatibilities, with entries being the calculated score associated with each assignment of student grouping to advisor grouping (calculated as described in the Model section); the other holds the allocations, the number of students assigned from each student grouping to each advisor grouping.

Once the table is created and filled with the appropriate calculated values, it is used to write our linear program. The variables are the number of students assigned from each student group to each advisor. A constraint is formed for each row and column. For each student group, the total number of students assigned from that group to any advisor groups must be strictly equal to the number of students in the grouping (that is, every student must be assigned, and no more students may be assigned than are available). Similarly, for each advisor group, the number of students assigned from any groups to that advisor group must be less than or equal to the total capacity (that is, no advisor group can receive more students than they can handle, but they can receive less than their capacity). Additional constraints are that each variable must remain non-negative and integer, which represent the intuitive ideas that a negative number of students cannot be assigned anywhere and that a half of a student cannot be assigned anywhere.

These constraints are written by the program to a text file by the writeLPFile() method, in the format specified by the GLPK, that the LP solver will read. Along with these constraints is our objective function, the maximization of the total score of all assignments (determined by multiplying the score of each assignment by the number of students allocated to that assignment and summing all such products). GLPSOL is invoked via command line and the results of the linear optimization are stored in a file. Optionally, sensitivity analysis for the optimization can also be recorded. Our program then looks at the results file and uses the optimal assignment information to recommend assignments. The linear program results indicate the number of students from each student grouping to assign to each professor. The program asks the user whether they would like to view the raw number assignments or if they would like specific assignments to be made for them. Should the user indicate that they prefer the automated assignments, the program randomly assigns the appropriate number of students from each section to the appropriate advisor. This random assignment is feasible and reasonable because of our simplification that students within the same major with the same level of TC can be equally well advised by any particular advisor and that advisors with the same level of training and in the same major can equally well advise any particular student. Regardless of the user's choice, the results are displayed and saved to a file for future use.

**4 Results**

We have developed software that, given aggregated data on an incoming class of students, can automatically assign these students to appropriate advisors in a way that maximizes total compatibility. In order to automate the pairing process we have worked with the CAS to quantify the existing questionnaire. With additional work on defining TC, our software is capable of interacting with the new online format of the questionnaire. This software will greatly reduce the amount of manual work that has been used in the past as well as decrease the amount of time needed to complete the assignment process.

**4.1 Post-optimality Analysis**

We were not able to obtain data from the CAS with previous student classes or the future incoming class, so we took data from the 2012 CIRP Freshman Survey for Southwestern to be a good estimate of incoming freshman. The survey sample size has 337 students which is more than 75% of an average incoming class, so it is a good representation of the class. We took the percentages of student preferences on major and expanded them to a class size of 447. We assume 39 transfer students and 10 international students. With the limited information about TC levels we had, we assumed an average class would have TC ranges of: 30% Low, 50% Middle, and 20% High. The data was parsed by our program from an Excel spreadsheet and ran. We also tested how the program handled different types of classes by changing the sample class to have High TC with TC ranges of: 10% Low, 60% Middle, and 30% High. We then tested a class size with double the amount of students interested in being a Biology major (~30%). This is already a popular major that CAS has a hard time sorting students to their ideal match. Choosing this class size shows how our program handles a more pronounced problem in this area of too many students in one major. We assume a fixed uniform capacity for every advisor, and we tested this capacity at three and at five.

We evaluated the assignment compatibility of our program with two different metrics. One being the percent of students assigned the advisor with the top compatibility score (as defined above) out of the total number of students. The other being the percentage of the compatibility score out of the max possible compatibility score. Max possible defined as every student having the top compatibility score or number of decided students\*8 + undecided\*5.

Table 1 shows the results of these three class sizes at the different capacity loads of five and three. Looking at our estimate for an average class with a capacity of five our program assigns 73% of students to top match and 61% with a capacity of three. The percentage of max scores being 93.37% and 86.57% respectively. The percentage of max score is higher than the percentage of students with their top match because the program assigns many students to their next best match (division, department). It also changes less than the percentage of students at top match because the program is always seeking the most ideal matches. Advisor capacity proves to be a substantial limiting factor. Comparing the average class to the High TC class finds that at an advisor capacity of five the percentages are almost identical. This reveals that at this capacity nearly all of the trained faculty are assigned advisees. This is expected since we have 50%of students (~223) that need a trained advisor and 60% of advisors (~100) that are trained. When the capacity is lowered to three the compatibility metrics are drastically lower than the average class, once again revealing that capacity is a very big limitation. Examining the Biology overloaded class surprisingly reveals that the program handles the assignment well with only a 6% drop from students at their top match at a capacity of five. With a capacity of three the Biology class has a higher compatibility score than that of the High TC class at the same capacity. So with a smaller capacity, the advisor supply is more able to handle a class with High TC than a class skewed towards one major.

**5 Conclusions and Recommendations**

The current implementation of the program works as we have defined. It is expandable to account for different sets of students and advisors. To fully integrate the program into a usable system for CAS, further work must be done by a computer science student or team. Ideally, it should work seamlessly with files CAS already keeps track of and the Google form questionnaire incoming classes are required to fill out. The current program assumes every advisor has the same capacity. This needs be changed to accommodate for each advisors capacity. It should read an input file that lists each advisor with their corresponding capacities, majors, and level of training. The program can also be expanded to require a minimum amount of advisees assigned for each advisor. Currently, the program reads TC as a number and sorts it into three distinct levels (low, middle, and high). Additional code must be written to read the Google form questionnaire and further translate the still qualitative questions into a numerical value, while working very closely with CAS to establish criteria and point values for the various TC. The TC levels, along with other values can and should be further defined.

To ensure the program is working as intended and making the best compatibility assignments the constraints should be optimized. Multiple sets of student data must be run through the program to fine tune the specific values. (TC, TC levels, and weight of TC, match in major, department division). Past student data should be run, and the program’s assignments compared to the previous manual method. Then if the data is available, analyze the students that had changed advisors and which advisors they changed to to see if our program may have made the more ideal match. Also, the program can be optimized by running it on future incoming classes or testing it on example fictitious classes.

We also believe advising would benefit from some form of assessment that evaluates what advisors and advisees think about the advising process. How they assess the compatibility, what works well, and what should be changed.

Our project is beneficial in that it automates the pairing process (increasing efficiency), clarifies existing variables used by the CAS (articulates and documents the process), and produces a system that outputs quantifiable feedback on each incoming class’s advising match-ups (allows for assessment). Implementation of our program will ease stress on CAS, while providing better matches for advisors and advisees.

# 6 Bibliography

"2012 CIRP Freshman Survey Institutional Profile Reports.": 8-11. *Southwestern University*.

Web. 27 Apr. 2014.

# 7 Appendix

#### Figure 1

#### Table 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Average |  | High TC |  | Bio Overload |  |
| Advisor Capacity | 5 | 3 | 5 | 3 | 5 | 3 |
| Students With Top Match | 73% | 61% | 73% | 51% | 67% | 54% |
| Percentage of max score | 93.37% | 86.57% | 93.08% | 81.01% | 90.16% | 84.59% |

#### 

# Program that parses input data, calculates assignment scores, and writes LP file for GLPSOL :

import java.io.\*;

import java.util.\*;

public class Main {

//Class variables. Do not touch

public static ArrayList<Major> majors=new ArrayList<Major>();

public static ArrayList<String> majorNames=new ArrayList<String>();

public static ArrayList<StudentGroup> studentGroups=new ArrayList<StudentGroup>();

public static ArrayList<AdvisorGroup> advisorGroups=new ArrayList<AdvisorGroup>();

public static String[][] variableNames;

public static int[][] scores;

//Program parameters. May be adjusted

public static int TC\_THRESHOLD=2;

public static int CAC\_TC\_THRESHOLD=6;

public static int POINTS\_FOR\_MDD=1;

public static int POINTS\_FOR\_TRAINING=5;

public static int FIXED\_UNIFORM\_CAPACITY=5;

public static void main(String[] args) throws IOException{

readInMajors();

createGroups();

readInStudents();

readInAdvisors();

updateGroups();

createVariablesAndscores();

writeLPFile();

}

//Read from file the list of majors and their corresponding divisions

public static void readInMajors() throws IOException{

File majorsFile=new File("MajorsList.txt");

Scanner reader=new Scanner(majorsFile);

while(reader.hasNextLine()){

String currentLine=reader.nextLine();

StringTokenizer tokenizer=new StringTokenizer(currentLine,",");

if(tokenizer.countTokens()!=3)

throw new IllegalArgumentException("Error in format of majors file");

Major major=new Major(tokenizer.nextToken(),tokenizer.nextToken(),tokenizer.nextToken());

majors.add(major);

majorNames.add(major.name);

}

reader.close();

}

//Based on the majors, create student groups for each

public static void createGroups(){

for(Major major:majors){

if(!major.name.equals("Transfer")&&!major.name.equals("Intnl")){

StudentGroup lowTC=new StudentGroup(major,false,false,false,false);

studentGroups.add(lowTC);

StudentGroup highTC=new StudentGroup(major,true,false,false,false);

studentGroups.add(highTC);

StudentGroup veryHTC=new StudentGroup(major,false,false,false,true);

studentGroups.add(veryHTC);

StudentGroup transfer=new StudentGroup(major,true,true,false,false);

studentGroups.add(transfer);

StudentGroup intnl=new StudentGroup(major,true,false,true,false);

studentGroups.add(intnl);

StudentGroup transferIntnl=new StudentGroup(major,true,true,true,false);

studentGroups.add(transferIntnl);

AdvisorGroup noTraining=new AdvisorGroup(false);

noTraining.addMajor(major);

advisorGroups.add(noTraining);

}

AdvisorGroup hasTraining=new AdvisorGroup(true);

hasTraining.addMajor(major);

advisorGroups.add(hasTraining);

}

}

//Read from file the list of students and assigns them to a group based on major and TC's

public static void readInStudents() throws IOException{

File studentFile=new File("StudentList.txt");

Scanner reader=new Scanner(studentFile);

while(reader.hasNextLine()){

String currentLine=reader.nextLine();

StringTokenizer tokenizer=new StringTokenizer(currentLine,",");

String name=tokenizer.nextToken()+tokenizer.nextToken();

if(name.contains("-"))

name=name.replace("-","");

String major=tokenizer.nextToken().replace(" ","");

String isTransfer=tokenizer.nextToken().toLowerCase();

String isIntnl=tokenizer.nextToken().toLowerCase();

int numTC=Integer.parseInt(tokenizer.nextToken());

boolean highTC=false,veryHighTC=false;

if(numTC>TC\_THRESHOLD)

highTC=true;

if(numTC>CAC\_TC\_THRESHOLD)

veryHighTC=true;

for(StudentGroup sgroup:studentGroups){

if(sgroup.major.name.equals(major)){

if(isIntnl.contains("y")&&isTransfer.contains("y")){

if(sgroup.isIntnl&&sgroup.isTransfer){

sgroup.addStudent(name);

break;

}

continue;

}

if(isIntnl.contains("y")){

if(sgroup.isIntnl){

sgroup.addStudent(name);

break;

}

continue;

}

if(isTransfer.contains("y")){

if(sgroup.isTransfer){

sgroup.addStudent(name);

break;

}

continue;

}

if(veryHighTC){

if(sgroup.veryHighTC){

sgroup.addStudent(name);

break;

}

continue;

}

if(sgroup.highTC==highTC){

sgroup.addStudent(name);

break;

}

}

}

}

}

//Read from file the list of advisors

public static void readInAdvisors() throws IOException{

File facultyFile=new File("FacultyList.txt");

Scanner reader=new Scanner(facultyFile);

while(reader.hasNextLine()){

String currentLine=reader.nextLine();

StringTokenizer tokenizer=new StringTokenizer(currentLine,",");

String name=tokenizer.nextToken();

if(name.contains("-"))

name=name.replace("-","");

String major\_s=tokenizer.nextToken();

String hasTrainingString=tokenizer.nextToken();

boolean hasTraining=false;

if(hasTrainingString.equals("1"))

hasTraining=true;

if(major\_s.contains("/")){

StringTokenizer majorTokenizer=new StringTokenizer(major\_s,"/");

ArrayList<Major> m=new ArrayList<Major>();

for(int i=0;i<majorTokenizer.countTokens();i++){

String majorString=majorTokenizer.nextToken();

if(!majorNames.contains(majorString))

throw new IllegalArgumentException("Unrecognized major for advisor "+name);

m.add(majors.get(majorNames.indexOf(majorString)));

}

boolean added=false;

for(AdvisorGroup agroup:advisorGroups){

if(agroup.majors.equals(m)&&agroup.hasTraining==hasTraining){

agroup.addAdvisor(name);

added=true;

break;

}

}

if(!added){

AdvisorGroup newGroup=new AdvisorGroup(hasTraining);

for(Major major:m){

newGroup.addMajor(major);

}

newGroup.addAdvisor(name);

advisorGroups.add(newGroup);

}

}

else{

if(!majorNames.contains(major\_s))

throw new IllegalArgumentException("Unrecognized major for advisor "+name);

ArrayList<Major> m=new ArrayList<Major>();

m.add(majors.get(majorNames.indexOf(major\_s)));

for(AdvisorGroup agroup:advisorGroups){

if(agroup.majors.equals(m)&&agroup.hasTraining==hasTraining){

agroup.addAdvisor(name);

break;

}

}

}

}

reader.close();

}

//Delete any groups that had low students added

public static void updateGroups(){

for(int i=0;i<studentGroups.size();i++){

if(studentGroups.get(i).numOfStudents==0){

studentGroups.remove(i);

i--;

}

}

for(int i=0;i<advisorGroups.size();i++){

if(advisorGroups.get(i).numOfAdvisors==0){

advisorGroups.remove(i);

i--;

}

}

}

//Create variable names and calculate associated scores based on group/advisor compatibilities

public static int[][] createVariablesAndscores(){

variableNames=new String[studentGroups.size()][advisorGroups.size()];

scores=new int[studentGroups.size()][advisorGroups.size()];

for(int row=0;row<variableNames.length;row++){

for(int col=0;col<variableNames[row].length;col++){

StudentGroup sgroup=studentGroups.get(row);

String variableName=sgroup.major.name;

if(sgroup.isIntnl)

variableName+="Intnl";

if(sgroup.isTransfer)

variableName+="Transfer";

if(sgroup.veryHighTC)

variableName+="VH";

else if(sgroup.highTC)

variableName+="H";

else

variableName+="L";

variableName+="To"+advisorGroups.get(col).majorsNames;

if(advisorGroups.get(col).hasTraining)

variableName+="H";

else

variableName+="L";

variableNames[row][col]=variableName;

}

}

for(int row=0;row<variableNames.length;row++){

for(int col=0;col<variableNames[row].length;col++){

StudentGroup sgroup=studentGroups.get(row);

AdvisorGroup agroup=advisorGroups.get(col);

int maxScore=0;

for(Major major:agroup.majors){

int score=0;

if(major.equals(sgroup.major))

score+=POINTS\_FOR\_MDD;

if(major.department.equals(sgroup.major.department))

score+=POINTS\_FOR\_MDD;

if(major.division.equals(sgroup.major.division))

score+=POINTS\_FOR\_MDD;

//Special Cases

if(sgroup.major.name.equals("AnimalBehavior")&&(major.department.equals("Biology")||major.department.equals("Psychology")))

score+=POINTS\_FOR\_MDD;

if(sgroup.veryHighTC&&major.name.equals("AcademicSuccess"))

score+=POINTS\_FOR\_TRAINING;

if(sgroup.isIntnl&&major.equals("Intnl"))

score+=POINTS\_FOR\_TRAINING;

if(sgroup.isTransfer&&major.name.equals("Transfer"))

score+=POINTS\_FOR\_TRAINING;

if(score>maxScore)

maxScore=score;

}

//Special Cases

if(sgroup.highTC&&!agroup.hasTraining)

maxScore+=0;

else

maxScore+=POINTS\_FOR\_TRAINING;

scores[row][col]=maxScore;

}

}

return scores;

}

//Write glp file using variable names and corresponding scores.

public static void writeLPFile() throws IOException{

File glpFile=new File("advising.lp");

PrintWriter output=new PrintWriter(glpFile);

output.println("Maximize");

output.print("Compatibility:");

for(int row=0;row<scores.length;row++){

for(int col=0;col<scores[row].length;col++){

output.print(" "+scores[row][col]+" "+variableNames[row][col]);

if(col!=scores[row].length-1||row!=scores.length-1)

output.println(" +");

}

}

output.println();

output.println("\nsubject to");

for(int row=0;row<scores.length;row++){

StudentGroup sgroup=studentGroups.get(row);

output.print(sgroup.major.name);

if(sgroup.isIntnl&&sgroup.isTransfer)

output.println("IntnlTransferStudents");

else if(sgroup.isIntnl)

output.print("IntnlStudents:");

else if(sgroup.isTransfer)

output.print("TransferStudents:");

else if(sgroup.veryHighTC)

output.print("VHStudents:");

else if(sgroup.highTC)

output.print("HStudents:");

else

output.print("LStudents:");

for(int col=0;col<scores[row].length;col++){

output.print(" "+variableNames[row][col]);

if(col!=scores[row].length-1)

output.print(" +");

else

output.println(" = "+sgroup.numOfStudents);

}

}

for(int col=0;col<scores[0].length;col++){

output.print(advisorGroups.get(col).majorsNames);

if(advisorGroups.get(col).hasTraining)

output.print("HAdvCap:");

else

output.print("LAdvCap:");

for(int row=0;row<scores.length;row++){

output.print(" "+variableNames[row][col]);

if(row!=scores.length-1)

output.print(" +");

else

output.println(" <= "+advisorGroups.get(col).totalCapacity);

}

}

output.println();

output.println("bounds");

for(int row=0;row<scores.length;row++){

for(int col=0;col<scores[row].length;col++){

output.println(variableNames[row][col]+" >= 0");

}

}

output.println();

output.println("integer");

for(int row=0;row<scores.length;row++){

for(int col=0;col<scores[row].length;col++){

output.println(variableNames[row][col]);

}

}

output.println("\nEnd");

output.close();

}

public static int[][] getScoreTable() throws IOException{

readInMajors();

createGroups();

readInStudents();

readInAdvisors();

updateGroups();

return createVariablesAndscores();

}

}

# Program that parses GLPSOL results, asks user how he/she wants results displayed, and displays results accordingly:

import java.io.\*;

import java.util.ArrayList;

import java.util.Random;

import java.util.Scanner;

import java.util.StringTokenizer;

public class ReadResults {

public static ArrayList<Major> majors=new ArrayList<Major>();

public static ArrayList<String> majorNames=new ArrayList<String>();

public static ArrayList<StudentGroup> studentGroups=new ArrayList<StudentGroup>();

public static ArrayList<AdvisorGroup> advisorGroups=new ArrayList<AdvisorGroup>();

public static int[][] assignments;

public static int z=0;

public static void main(String[] args) throws IOException{

readInMajors();

createGroups();

readInStudents();

readInAdvisors();

updateGroups();

readResults();

System.out.println("Would you like the program to assign specific students?");

Scanner input=new Scanner(System.in);

if(input.next().toLowerCase().contains("n"))

reportGeneralResults();

else

assignSpecificStudents();

input.close();

assessResults();

}

//Read from file the list of majors and their corresponding divisions

public static void readInMajors() throws IOException{

File majorsFile=new File("MajorsList.txt");

Scanner reader=new Scanner(majorsFile);

while(reader.hasNextLine()){

String currentLine=reader.nextLine();

StringTokenizer tokenizer=new StringTokenizer(currentLine,",");

if(tokenizer.countTokens()!=3)

throw new IllegalArgumentException("Error in format of majors file");

Major major=new Major(tokenizer.nextToken(),tokenizer.nextToken(),tokenizer.nextToken());

majors.add(major);

majorNames.add(major.name);

}

reader.close();

}

//Based on the majors, create student groups for each

public static void createGroups(){

for(Major major:majors){

if(!major.name.equals("Transfer")&&!major.name.equals("Intnl")){

StudentGroup lowTC=new StudentGroup(major,false,false,false,false);

studentGroups.add(lowTC);

StudentGroup highTC=new StudentGroup(major,true,false,false,false);

studentGroups.add(highTC);

StudentGroup veryHTC=new StudentGroup(major,false,false,false,true);

studentGroups.add(veryHTC);

StudentGroup transfer=new StudentGroup(major,true,true,false,false);

studentGroups.add(transfer);

StudentGroup intnl=new StudentGroup(major,true,false,true,false);

studentGroups.add(intnl);

StudentGroup transferIntnl=new StudentGroup(major,true,true,true,false);

studentGroups.add(transferIntnl);

AdvisorGroup noTraining=new AdvisorGroup(false);

noTraining.addMajor(major);

advisorGroups.add(noTraining);

}

AdvisorGroup hasTraining=new AdvisorGroup(true);

hasTraining.addMajor(major);

advisorGroups.add(hasTraining);

}

}

//Read from file the list of students and assigns them to a group based on major and TC's

public static void readInStudents() throws IOException{

File studentFile=new File("StudentList.txt");

Scanner reader=new Scanner(studentFile);

while(reader.hasNextLine()){

String currentLine=reader.nextLine();

StringTokenizer tokenizer=new StringTokenizer(currentLine,",");

String name=tokenizer.nextToken()+tokenizer.nextToken();

if(name.contains("-"))

name=name.replace("-","");

String major=tokenizer.nextToken().replace(" ","");

String isTransfer=tokenizer.nextToken().toLowerCase();

String isIntnl=tokenizer.nextToken().toLowerCase();

int numTC=Integer.parseInt(tokenizer.nextToken());

boolean highTC=false,veryHighTC=false;

if(numTC>Main.TC\_THRESHOLD)

highTC=true;

if(numTC>Main.CAC\_TC\_THRESHOLD)

veryHighTC=true;

for(StudentGroup sgroup:studentGroups){

if(sgroup.major.name.equals(major)){

if(isIntnl.contains("y")&&isTransfer.contains("y")){

if(sgroup.isIntnl&&sgroup.isTransfer){

sgroup.addStudent(name);

break;

}

continue;

}

if(isIntnl.contains("y")){

if(sgroup.isIntnl){

sgroup.addStudent(name);

break;

}

continue;

}

if(isTransfer.contains("y")){

if(sgroup.isTransfer){

sgroup.addStudent(name);

break;

}

continue;

}

if(veryHighTC){

if(sgroup.veryHighTC){

sgroup.addStudent(name);

break;

}

continue;

}

if(sgroup.highTC==highTC){

sgroup.addStudent(name);

break;

}

}

}

}

}

//Read from file the list of advisors

public static void readInAdvisors() throws IOException{

File facultyFile=new File("FacultyList.txt");

Scanner reader=new Scanner(facultyFile);

while(reader.hasNextLine()){

String currentLine=reader.nextLine();

StringTokenizer tokenizer=new StringTokenizer(currentLine,",");

String name=tokenizer.nextToken();

if(name.contains("-"))

name=name.replace("-","");

String major\_s=tokenizer.nextToken();

String hasTrainingString=tokenizer.nextToken();

boolean hasTraining=false;

if(hasTrainingString.equals("1"))

hasTraining=true;

if(major\_s.contains("/")){

StringTokenizer majorTokenizer=new StringTokenizer(major\_s,"/");

ArrayList<Major> m=new ArrayList<Major>();

for(int i=0;i<majorTokenizer.countTokens();i++){

String majorString=majorTokenizer.nextToken();

if(!majorNames.contains(majorString))

throw new IllegalArgumentException("Unrecognized major for advisor "+name);

m.add(majors.get(majorNames.indexOf(majorString)));

}

boolean added=false;

for(AdvisorGroup agroup:advisorGroups){

if(agroup.majors.equals(m)&&agroup.hasTraining==hasTraining){

agroup.addAdvisor(name);

added=true;

break;

}

}

if(!added){

AdvisorGroup newGroup=new AdvisorGroup(hasTraining);

for(Major major:m){

newGroup.addMajor(major);

}

newGroup.addAdvisor(name);

advisorGroups.add(newGroup);

}

}

else{

if(!majorNames.contains(major\_s))

throw new IllegalArgumentException("Unrecognized major for advisor "+name);

ArrayList<Major> m=new ArrayList<Major>();

m.add(majors.get(majorNames.indexOf(major\_s)));

for(AdvisorGroup agroup:advisorGroups){

if(agroup.majors.equals(m)&&agroup.hasTraining==hasTraining){

agroup.addAdvisor(name);

break;

}

}

}

}

reader.close();

}

//Delete any groups that had low students added

public static void updateGroups(){

for(int i=0;i<studentGroups.size();i++){

if(studentGroups.get(i).numOfStudents==0){

studentGroups.remove(i);

i--;

}

}

for(int i=0;i<advisorGroups.size();i++){

if(advisorGroups.get(i).numOfAdvisors==0){

advisorGroups.remove(i);

i--;

}

}

}

public static void readResults(){

try{

File results=new File("lp\_results.txt");

Scanner reader=new Scanner(results);

assignments=new int[studentGroups.size()][advisorGroups.size()];

for(int row=0;row<assignments.length;row++){

for(int col=0;col<assignments[row].length;col++){

while(reader.hasNext()){

String next=reader.next();

if(next.equals("Compatibility")){

reader.next();

z=reader.nextInt();

}

if(next.contains("LTo")||next.contains("HTo")){

reader.next();

assignments[row][col]=reader.nextInt();

break;

}

}

}

}

reader.close();

}

catch(IOException e){

System.out.println("Error reading file");

}

}

public static void reportGeneralResults(){

try{

File results=new File("assignment\_results.txt");

PrintWriter output=new PrintWriter(results);

for(int row=0;row<assignments.length;row++){

for(int col=0;col<assignments[row].length;col++){

if(assignments[row][col]==0)

continue;

StudentGroup sgroup=studentGroups.get(row);

output.print("Send "+assignments[row][col]+" student(s) from the "+sgroup.major.name+" major ");

if(sgroup.veryHighTC)

output.print("with very high transitional challenges");

else if(sgroup.highTC)

output.print("with high transitional challenges ");

else

output.print("with low transitional challenges ");

AdvisorGroup agroup=advisorGroups.get(col);

output.print("to advisors in the "+agroup.majorsNames);

if(agroup.hasTraining)

output.println(" major(s) who has training");

else

output.println(" major(s) who doesn't have training");

}

}

output.close();

}

catch(IOException e){

System.out.println("Error writing results to file");

}

}

public static void assignSpecificStudents(){

try{

File results=new File("assignment\_results.txt");

PrintWriter output=new PrintWriter(results);

for(int col=0;col<assignments[0].length;col++){

AdvisorGroup agroup=advisorGroups.get(col);

for(int row=0;row<assignments.length;row++){

StudentGroup sgroup=studentGroups.get(row);

for(int index=0;index<assignments[row][col];index++){

output.println("Assign "+sgroup.students.get(0)+" to "+agroup.advisors.get(index%agroup.numOfAdvisors));

sgroup.students.remove(0);

}

}

}

output.close();

}

catch(IOException e){

System.out.println("Error writing results to file");

}

}

public static void assessResults() throws IOException{

int numHappy1=0,num10=0,num5=0,num8=0;

int[][] scores=Main.getScoreTable();

for(int i=0;i<scores.length;i++){

for(int j=0;j<scores[0].length;j++){

StudentGroup sgroup=studentGroups.get(i);

if(sgroup.veryHighTC||sgroup.isIntnl||sgroup.isTransfer){

if(scores[i][j]==10)

numHappy1+=assignments[i][j];

num10+=assignments[i][j];

}

else if(sgroup.major.name.equals("Undecided")){

if(scores[i][j]>=5)

numHappy1+=assignments[i][j];

num5+=assignments[i][j];

}

else{

if(scores[i][j]>=8)

numHappy1+=assignments[i][j];

num8+=assignments[i][j];

}

}

}

int numStudents=0;

for(StudentGroup sgroup:studentGroups){

numStudents+=sgroup.numOfStudents;

}

System.out.println(numHappy1\*100/numStudents+"% of students with top choice.");

System.out.println("10: "+num10+" 8: "+num8+" 5: "+num5);

System.out.println("Max possible score: "+(10\*num10+8\*num8+5\*num5));

System.out.println("Percentage of max score achieved: "+(double)z/(10\*num10+8\*num8+5\*num5)\*100);

}

}

**List of Majors and Their Associated Departments and Divisions**

Accounting,Economics&Business,SSD

AnimalBehavior,Biology/Psychology,NSD

Anthropology,Sociology&Anthropology,SSD

Art,Art,SSFA

ArtHistory,ArtHistory,SSFA

Biochemistry,Chemistry&Biochemistry,NSD

Biology,Biology,NSD

Business,Economics&Business,SSD

Chemistry,Chemistry&Biochemistry,NSD

Chinese,Languages,HUM

Classics,Classics,HUM

Communication,Comm.Studies,HUM

ComputerScience,Math&ComputerScience,NSD

Economics,Economics&Business,SSD

Education,Education,SSD

English,English,HUM

EnvironmentalStudies,EnvironmentalStudies,SSD

FeministStudies,FeministStudies,SSD

French,Languages,HUM

German,Languages,HUM

Greek,Classics,HUM

History,History,HUM

Kinesiology,Athletics,NSD

Latin,Classics,HUM

LatinAmericanStudies,Spanish,HUM

Math,Math&ComputerScience,NSD

Music,Music,SSFA

Philosophy,Philosophy,HUM

Physics,Physics,NSD

PoliticalScience,PoliticalScience,SSD

Psychology,Psychology,SSD

Religion,Religion,HUM

Sociology,Sociology&Anthropology,SSD

Spanish,Languages,HUM

Theatre,Theatre,SSFA

AcademicSuccess,N/A,N/A

Admissions,N/A,N/A

CareerDevelopment,N/A,N/A

DeanofStudents,N/A,N/A

InformationTechnology,N/A,N/A

Library,N/A,N/A

Provost,N/A,N/A