AC 2011-1535: AN OPTIMIZATION ROUTINE FOR ASSIGNING STUDENTS TO CAPSTONE PROJECT GROUPS

Peter L Schmidt, University of North Carolina, Charlotte

Peter L. Schmidt received his bachelor's degree in mechanical engineering from the University of Louisville, a master's degree in mechanical engineering from the Rose-Hulman Institute of Technology and his doctorate degree in mechanical engineering from Vanderbilt University. He is currently an assistant professor at the University of North Carolina at Charlotte. He has served as a research associate and as an instructor at Vanderbilt University. He has also worked at the Naval Surface Warfare Center in Crane, Indiana; at Precision Rubber, now part of Parker Hannifin in Lebanon, Tennessee; for CDAI in Atlanta, Georgia and at UTC / Carrier in Lewisburg, Tennessee. Dr. Schmidt is a member of the ASEE and a licensed professional engineer in Tennessee and Georgia. He is also a member of ASME, ASHRAE, ASA and INCE. Dr. Schmidt's research interests include aeroacoustics and ultrasonics, and has authored several journal and conference papers on these subjects.

Daniel Hoch, University of North Carolina, Charlotte
Nabila A. Bousaba, University of North Carolina, Charlotte

Nabila (Nan) BouSaba, University of North Carolina, Charlotte Nabila (Nan) BouSaba is a faculty associate in the Electrical and Computer Engineering Department at the University of North Carolina in Charlotte. Nan Earned her BS in Electrical Engineering (1982), and a Master Degree in Electrical Engineering (1986) from North Carolina A&T State University. Prior to her current position at UNC- Charlotte, Nan worked for IBM (15 years) and Solectron (8 years) in the area of test development and management. She teaches the senior design course for the Electrical and computer sections and Basic Electrical Circuit course.

William F. Heybruck, University of North Carolina, Charlotte

Bill received is BSEE degree from Merrimack College in North Andover MA, a Masters in Computer Science from Union College in Schenectady NY and more recently his Ph.D. in EE from UNC Charlotte in 2001. He was with IBM for 32 years when he retired as a Hard Disk Drive Consultant when Hitachi bought his division. Prior to that, he was a consulting engineer for test technology, a wireless consultant and a Product Development Manager in Printer Development. He worked for Hitachi Global Storage Technology for 5 years as an expert on Micro hard disk drives before coming to UNC Charlotte as Director of the Industrial Solutions Lab.

Deborah L Sharer, University of North Carolina, Charlotte
Valentina Cecchi, The University of North Carolina at Charlotte

Valentina Cecchi is originally from Rome, Italy. She attended Drexel University in Philadelphia, PA, where she completed B.S., M.S., and Ph.D. degrees in Electrical Engineering in 2005, 2007, and 2010, respectively. She joined UNC Charlotte in 2010 as Assistant Professor of Electrical Engineering and researcher in the Energy Production and Infrastructure Center (EPIC).

S. Gary Teng, The University of North Carolina at Charlotte

Dr. S. Gary Teng is Professor and Director of Systems Engineering & Engineering Management Program and Center for Lean Logistics & Engineered Systems at the University of North Carolina at Charlotte. He holds B.E., M.S., and Ph.D. degrees in Industrial Engineering. Dr. Teng is a Professional Engineer in the State of Wisconsin and an ASQ-certified Quality Engineer and Reliability Engineer. His research interests are in engineering system design and analysis, lean systems design & implementation, Lean logistics and transportation systems, supply chain management, healthcare system enhancement, and quality and reliability engineering. Dr. Teng has been appointed by North Carolina Governor Bev Perdue to serve on Governor's Logistics Task Force. He is also serving as an advisor on the ad hoc Logistics Advisory Group for Charlotte Regional Logistics Network and on the board of Logistics Alliance of the Carolinas.

Elizabeth Sharer, Francis Marion University

Introduction

At the beginning of the first semester of the two-semester sequence a list of potential projects are offered to students in a multidisciplinary capstone course at The University of North Carolina at Charlotte via the course website. Each project is accompanied by a description submitted by the sponsoring organization or individual, and has been vetted by the course's faculty committee before display. These project descriptions include a suggested staffing level for the project, broken-down by major area of study (i.e., Mechanical, Electrical, Computer, Civil, etc.). Unless specified, there is no distinction between Engineering Sciences and Engineering Technology disciplines. The faculty committee associated with this course has decided that student choice in project assignments is a critical component to project success, and, therefore, enrolled students are granted the opportunity to rank their top three project choices. These student rankings are used, in association with the expertise requirements of the project, to staff projects each semester. In addition, the faculty has implemented a system where a student's cumulative GPA is used as a weighting factor. If two students from the same technical discipline rank projects in the same order, the student with the higher GPA would be given preference in obtaining their highest ranked project choice that is still open for staffing. The capstone course features projects that are sponsored internally as well as externally. Externally sponsored projects serve a dual purpose to the program and to the university at large. These projects provide students with the experience of working with established engineers. They also provide a marketing opportunity for the College of Engineering (COE), as well as the capstone design program. In the last four years, course population has increased from 60 to 265. The solution described herein for the student assignment problem allows projects to be staffed with students using a weighted coefficient for each student/project combination. This course begins with an event where project representatives are present to answer student queries regarding project specifics and expectations. The course timetable is such that project assignments need to be made quickly, so that an initial planning meeting with student teams, faculty supervisors and project sponsors can take place in week three of the semester.

When the entire class size was 30 students, the project staffing process was accomplished by hand. Now that the student population has exceeded 200, this is no longer an option, either from an accuracy or execution time standpoint. The vast majority of departments and programs have all students participate in the COE senior design, with an emphasis on drastic reduction of self-defined and self-funded projects. This is still a possibility, but all students are required to work on teams and a detailed project proposal must be submitted and before the semester begins. An automated method was desired, and its development and testing is the subject of this work.

Implementation of the solution

Since the staffing process represents a multivariate optimization with a large search space (more than 200 students and 70 projects in Fall 2010), a Genetic Algorithm (GA) was chosen as the tool of choice to execute the problem.

A GA is a problem solving technique that uses the concepts of evolution and heredity to produce quality solutions to complex problems that typically have enormous search spaces and are therefore difficult to solve. GAs have proved to be very helpful in solving complex, combinatorial problems. Reference [1] is a survey of various implementations of GAs in solving complex, combinatorial production and operations management problems. A well designed GA allows for the efficient and effective exploration and exploitation of the problem's search space of feasible solutions in an effort to identify the global optimal, or near optimal, solution to difficult problems [2].

A GA creates and manipulates a group of possible solutions referred to as a population. Each possible solution within the population is called an individual. The population undergoes change throughout the run of the GA thereby evolving the individuals toward a best solution. Within the GA, the population loops through a series of processes a number of times; each executed loop is known as a generation. These processes include an evaluation process, an alteration process, and a selection process. These processes may occur in various orders; however, each is required at each generation. [2]

The evaluation process uses an evaluation function that assesses the relative fitness of each individual of the population at each generation. In addition, at each generation a number of individuals are subjected to some form of change. These alterations are manifested through the use of genetic operators. Genetic operators can be either mutation operators, which introduce small changes within a single individual, or crossover operators, which cut and paste different parts from two or more individuals together in order to create new individuals called offspring. The probability of an individual experiencing some form of transformation within any given generation is subject to the predefined parameters of the probability of mutation, and/or the probability of crossover. Through this process, some, or all, of the individuals are altered and used to create a new population for the next generation. Finally, the GA uses the evaluated fitness of each individual to promote the survival of the best individuals to the next generation. This use of selective pressure encourages the population to converge to a quality solution. The GA will run for a predetermined maximum number of generations or until some specified terminating condition is met. [3]

Each GA is unique in its design with regard to several important elements. Some of these elements include data structure, genetic operators, method for creating the initial population, constraint handling techniques, evaluation function, selection method, generational policy, parameters, and terminating conditions. Parameters include population size, maximum number of generations, probability of mutation, and/or the probability of crossover. However, regardless of the differences, all GAs attempts to evolve the individuals within the population through the

use of genetic operators and selective pressures to converge at a suitable solution to complex problems. [3]

The inputs for the GA used in this solution are given as:

Population Size: The number of individuals in each generation, during the execution of the algorithm.

Crossover 1 Probability: The probability that crossover operator #1 will be applied. If a randomly generated number is less than or equal to the crossover probability value, then crossover #1 will be performed.

Crossover 2 Probability: The probability that crossover operator #2 will be applied with the same rules as crossover operator #1.

Mutation 1 Probability: The probability that mutation operator #1 will be applied. If a randomly generated number is less than or equal to the mutation 1 probability value, then mutation #1 will be performed.

Mutation 2 Probability: The probability that mutation operator #2 will be applied with the same rules as mutation operator #1.

Chance: Applies to crossover #1, crossover #2, mutation #1, and mutation #2 operators. This value sets the number of attempts to perform each operator.

Maximum number of generations: This value sets the number of search iterations the algorithm will run. This is the number of generations a population of individual solutions will evolve. As the algorithm runs, the solutions evolve, thereby randomly sampling the problem's search space. For our implementation, the individual solution identified with the largest fitness value during the run of the GA is given as the "best" solution.

The development of a GA is complex. Generally, the larger the search space (in this case, the more student/project combinations), the larger the population size and maximum number of generations required in order to adequately examine the problem's search space. For the student assignment problem a population size of 20 to 40 was theorized as sufficient, along with a maximum number of generations below 200.

The data structure employed to solve the student assignment problem is a simple two-dimensional binary array as shown in Figure 1. Each student is assigned to exactly one project; however, each project may have many students. Each cell in the two-dimensional array represents a unique student/project combination. If a student is assigned to a given project there will be a one entered in that student/project index, otherwise the value will be zero.

Individual 1						
	Project 1	Project 2	Project 3	...	Project *m*	
Student 1	{0,1}	{0,1}	{0,1}	...	{0,1}	$\sum = 1$
Student 2	{0,1}	{0,1}	{0,1}	...	{0,1}	$\sum = 1$
Student 3	{0,1}	{0,1}	{0,1}	...	{0,1}	$\sum = 1$
:	{0,1}	{0,1}	{0,1}	...	{0,1}	$\sum = 1$
Student *n*	{0,1}	{0,1}	{0,1}	...	{0,1}	$\sum = 1$

Figure 1. Basic Data Structure for Individual Solutions.

As discussed, each project is assigned a weight based on its priority. The actual weight is not important, and, in fact, the spread of weights can be whatever is desired, the important thing is that the highest priority projects have the highest weights and the lowest priority projects have the lowest weights.

Furthermore, each student/project combination is given a weighted coefficient. The weighted coefficient is the product of the student's GPA, the choice weight based on the project choices made by the student, and the project weight. If a project is not one of a student's three top choices, the project's ranking is equal to one. Figure 2 illustrates how the student/project coefficient weights are calculated for each student/project combination. Figure 3 further illustrates that if a student/project combination is not selected than its cell value will be zero, and, therefore, its contribution to the overall individual's fitness value will be zero. If the student/project combination is selected, then the contribution to the overall individual's fitness value will be equal to the student/project cell's weight.

Individual 1					
	Project 1	Project 2	Project 3	...	Project *m*
Student 1	$w_{1,1}$ = Student 1's GPA * Student 1's Project Choice Weight * Project 1's weight	$w_{1,2}$	$w_{1,3}$	...	$w_{1,m}$
Student 2	$w_{2,1}$ = Student 2's GPA * Student 2's Project Choice Weight * Project 1's weight	$w_{2,2}$	$w_{2,3}$	...	$w_{2,m}$
Student 3	$w_{3,1}$ = Student 3's GPA * Student 3's Project Choice Weight * Project 1's weight	$w_{3,2}$	$w_{3,3}$	...	$w_{3,m}$
:	...	...	...	...	...
Student *n*	$w_{n,1}$ = Student *n* 's GPA * Student *n* 's Project Choice Weight * Project 1's weight	$w_{n,2}$	$w_{n,3}$	...	$w_{n,m}$

Figure 2. Calculation of Total Project Weight

Individual 1					
	Project 1	Project 2	Project 3	...	Project *m*
Student 1	{0,1} * $w_{1,1}$	{0,1} * $w_{1,2}$	{0,1} * $w_{1,3}$	...	{0,1} * $w_{1,m}$
Student 2	{0,1} * $w_{2,1}$	{0,1} * $w_{2,2}$	{0,1} * $w_{2,3}$	...	{0,1} * $w_{2,m}$
Student 3	{0,1} * $w_{3,1}$	{0,1} * $w_{3,2}$	{0,1} * $w_{3,3}$	...	{0,1} * $w_{3,m}$
:	...	...	...	...	...
Student *n*	{0,1} * $w_{n,1}$	{0,1} * $w_{n,2}$	{0,1} * $w_{n,3}$	...	{0,1} * $w_{n,m}$

*w represents the Student/Project weight

Figure 3. Weighted Data Structure

Once the student/project weights are calculated, assignment of students to projects (initially done semi-randomly) creates the first generation of individuals, each representing a feasible solution, and a corresponding of fitness values, which is the cumulative weight of all selected student/project combinations. The aggregated fitness value is used, in part, to determine which individuals "survive" to the next generation (i.e., the set of solutions used in the next iteration).

Projects with larger staffing requirements can produce a larger fitness value to the total individual's fitness value than that contributed by a project with smaller staff requirements. Therefore, the construction of this GA is such that large projects are given preference with respect to smaller ones for staffing, given an equal project priority. Small projects are often cancelled with extraneous personnel assigned to a larger project, rather than execution with a short staff. This solution facilitates rounding out group assignments in practice. Figure 4 illustrates the population of individuals, each student/project combination within each individual within the population will be assigned with a 0 or 1, where a 1 indicates that a student is assigned to a given project. A student is assigned to exactly one project. Each individual's fitness value is the cumulative product of the weighted coefficient for each selected student/project combination.

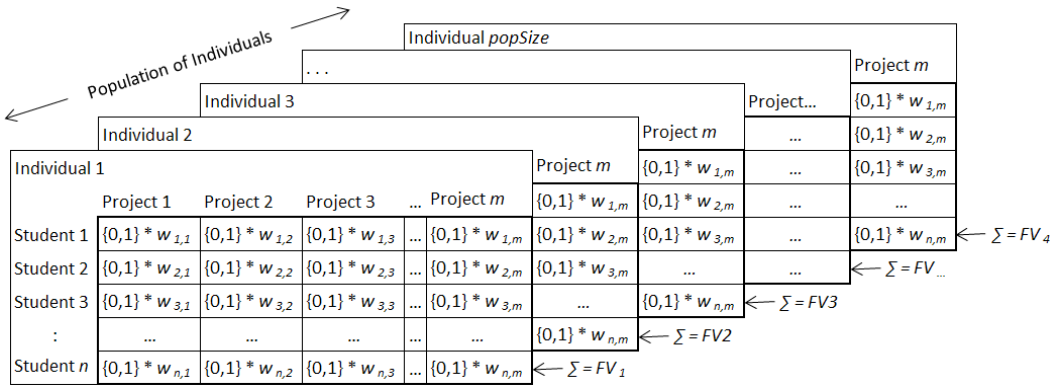

Figure 4. Calculation of Fitness Value

Once the initial generation of individual solutions has been created, the mechanism of the GA is engaged. Through two basic genetic operations, Crossover and Mutation, the solutions in a

generation's population randomly undergo augmentation and change, thereby creating new individual solutions called offspring. These genetic operators produce feasible solutions, each with its own fitness value. Just as in nature, individuals with genetic changes with larger fitness values have a higher probability of being selected for the next generation. At the end of each generation, there is a pool of individuals that is comprised of the individuals in the population at the beginning of the generation and all offspring that have been created during the run of the generation.

The selection process for populating each successive generation is completed through the use of a tournament selection process. In the implementation of the tournament selection process used in this GA, three individuals are randomly selected from the pool of individuals at the end of a generation; the individual with the largest fitness value is deemed the "winner" and is copied into the next generation's population. The selection process for the next generation's population also employees an elites strategy, which entails copying the best solution found to that point from the run of the algorithm into each generation. The tournament selection process is then run *popSize – 1*, where *popSize* is the constant population size maintained from generation to generation.

Of the two basic genetic operations, Crossover and Mutation, this GA implements two crossover operators, designated as Crossover 1 and Crossover 2, and two mutation operators, designated Mutation 1 and Mutation 2. Both crossover operators create two "offspring", or new individual solutions for possible inclusion in the next generation, by swapping a portion of each parent into each offspring. Both mutation operators create one "offspring", by making a small change in one "parent" solution. Figure 5 illustrates how Crossover 1 operator works. Here a "cut-point" corresponding to a student number is randomly selected, i.e., *randStudentNum*. The information from Student 1 to Student *randStudentNum-1* is copied from Parent 1 and Parent 2 into Offspring 1 and Offspring 2, respectively. Then the information from Student *randStudentNum* to Student *n*, where *n* is the total number of students, is copied from Parent 1 and Parent 2 into Offspring 2 and Offspring 1, respectively. This function perturbs the "student" information in the Parent individuals, thereby creating two new individuals, i.e., the row wise information encoded in contributing individuals.

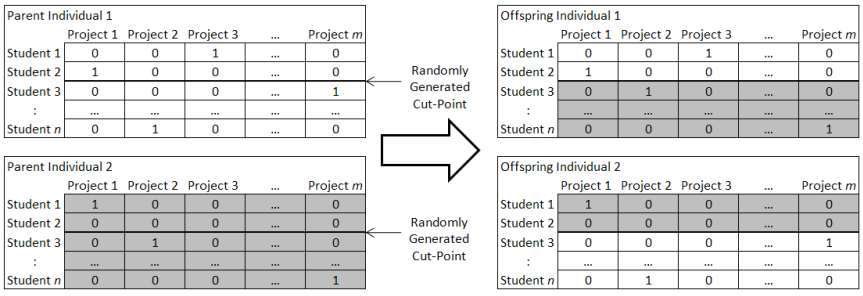

Figure 5. Offspring produced by the Crossover 1 Operator

For Crossover 2, again a "cut-point" is randomly selected. However, for this crossover operator the cut-point corresponds to a project number, i.e., *randProjectNum*. The information from Project 1 to Project *randProjectNum-1* is copied from Parent 1 and Parent 2 into Offspring 1 and Offspring 2, respectively. Then the information from Project *randProjectNum* to Project *m*, where *m* is the total number of projects, is copied from Parent 1 and Parent 2 into Offspring 2 and Offspring 1, respectively. This function perturbs the "project" information in the Parent individuals, thereby creating two new individuals, i.e., the column wise information encoded in contributing individuals, as shown in Figure 6.

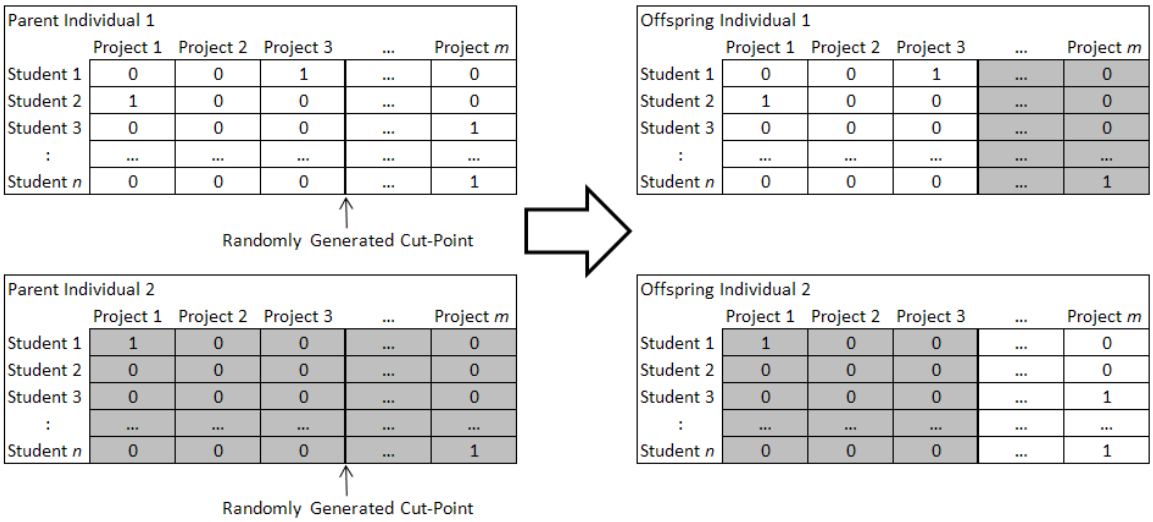

Figure 6. Offspring Produced by the Crossover 2 Operator

Rather than producing offspring through combination of genetic material, the mutation operator produces a changed individual by randomly changing only a small amount of information within one Parent individual. Mutation 1 generates two random numbers which correspond to two student numbers. The information for each of the two student numbers is then swapped, thereby producing a new individual for possible inclusion in the next generation of solutions. This process is illustrated in Figure 7.

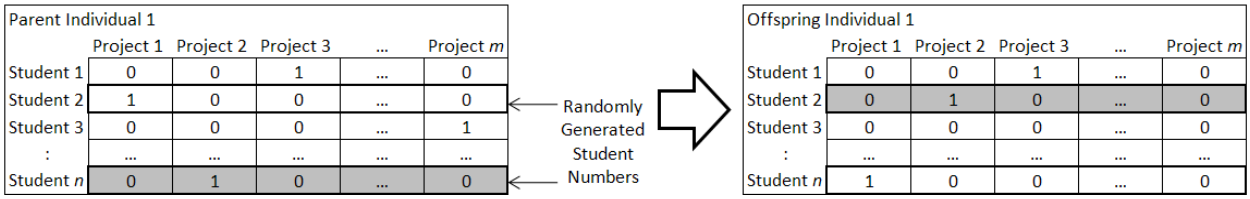

Figure 7. New individual produced by the Mutation 1 Operator

Mutation 2 is a more targeted change. A Parent individual is selected, and a single student/project allele is designated randomly for assignment. The student/project allele previously assigned is deleted, as shown in Figure 8.

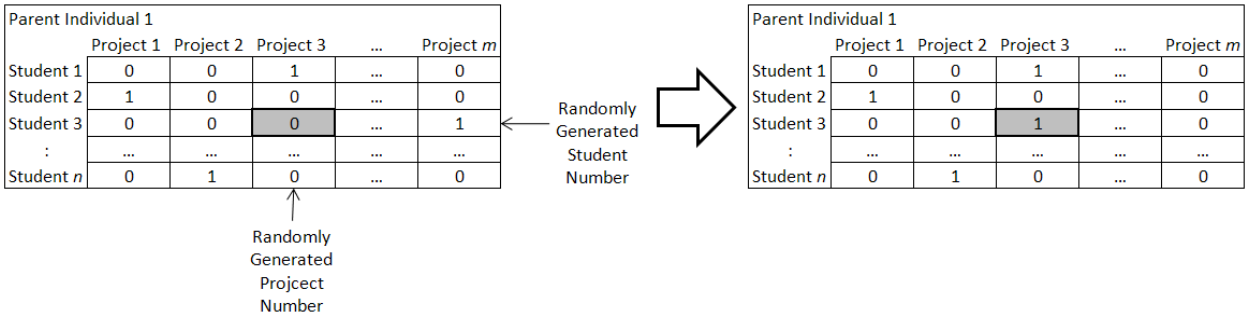

Figure 8. New individual produced by the Mutation 2 Operator

The GA application is designed so that any or all of the four genetic operators can be applied. For our initial tests, all four operators were used, where the probability of each (i.e., Crossover 1 Probability, Crossover 2 Probability, Mutation 1 Probability, and Mutation 2 Probability) where all set to 0.5. Here the associated probabilities indicate when the operator will be implemented. For this GA, for each genetic operator a random number is generated *Chance* times, where *Chance* is an additional input variable. If the random number generated is less than the genetic operator's probability, then the operator is implemented. For this GA, the *Chance* is the same for all genetic operators and can be changed. For the initial tests, *Chance* was set equal to the population size. The *Chance* parameter and the different genetic operator probabilities are a mechanism of how much perturbation is desired between generations. The program code for this GA is included as Appendix I.

Results

To test the algorithm, two data sets were used. The first data set consisted of 67 students and 24 projects. The number of individuals, the chance for genetic change and the maximum number of generations were varied to check for convergence of the method. The output of the solutions generated by the algorithm are shown in Table 1. These solutions were obtained with the following input parameters:

- Crossover 1 Probability = 0.5
- Crossover 2 Probability = 0.5
- Mutation 1 Probability = 0.5
- Mutation 2 Probability = 0.5

Table 1. Convergence Study

			Over 5 Runs			Student Choice (Average over 5 runs)				% Student Choice (Average over 5 runs)			
Population Size	Chance	Max # Generations	Maximum	Minimum	Average	1st	2nd	3rd	other	1st	2nd	3rd	other
10	10	20	84,345	81,420	81,420	37.2	12.4	4.6	12.8	55.5%	18.5%	6.9%	19.1%
20	20	20	83,627	80,955	82,374	36.6	11.4	6.4	12.6	54.6%	17.0%	9.6%	18.8%
30	30	20	84,275	81,836	82,654	38	12.6	5.8	10.6	56.7%	18.8%	8.7%	15.8%
40	40	20	84,753	82,099	83,382	35	13.4	5.4	13.2	52.2%	20.0%	8.1%	19.7%
50	50	20	83,478	81,002	82,506	38	13.2	5.4	10.4	56.7%	19.7%	8.1%	15.5%
10	10	40	84,415	82,560	83,615	34.6	12.2	6.2	14	51.6%	18.2%	9.3%	20.9%
20	20	40	84,729	81,519	83,675	37.4	13.6	5	11	55.8%	20.3%	7.5%	16.4%
30	30	40	84,109	79,795	82,645	38	12.8	6.2	10	56.7%	19.1%	9.3%	14.9%
40	40	40	84,941	79,790	82,250	38.6	11.8	5.4	11.2	57.6%	17.6%	8.1%	16.7%
50	50	40	84,441	81,908	83,343	38.6	13.2	5.6	9.6	57.6%	19.7%	8.4%	14.3%
10	10	60	91,274	80,293	84,568	33.2	12	5	16.8	49.6%	17.9%	7.5%	25.1%
20	20	60	85,366	82,274	83,825	38.4	13	5.8	9.8	57.3%	19.4%	8.7%	14.6%
30	30	60	89,536	82,599	84,680	35.4	13.4	5.6	12.6	52.8%	20.0%	8.4%	18.8%
40	40	60	85,906	81,079	82,873	37.2	12.6	4.6	12.6	55.5%	18.8%	6.9%	18.8%
50	50	60	84,774	80,061	82,514	37.4	12.4	5.2	12	55.8%	18.5%	7.8%	17.9%
10	10	80	86,391	82,192	83,915	38	12.2	5	11.8	56.7%	18.2%	7.5%	17.6%
20	20	80	84,131	81,311	83,387	37	13.2	5.6	11.2	55.2%	19.7%	8.4%	16.7%
30	30	80	87,774	80,760	83,278	37.2	11.8	6.2	11.8	55.5%	17.6%	9.3%	17.6%

40	40	80	84,848	81,001	83,207	37.6	12.8	5.8	10.8	56.1%	19.1%	8.7%	16.1%
50	50	80	84,707	81,306	82,720	39.2	12	6.4	9.4	58.5%	17.9%	9.6%	14.0%
10	10	100	85,454	82,622	83,864	33	12.8	4.2	17	49.3%	19.1%	6.3%	25.4%
20	20	100	87,243	81,541	84,472	38.6	12.6	6	9.8	57.6%	18.8%	9.0%	14.6%
30	30	100	84,831	79,614	83,216	39.4	12.2	6.4	9	58.8%	18.2%	9.6%	13.4%
40	40	100	85,684	83,450	84,415	37.4	13.2	5.8	10.6	55.8%	19.7%	8.7%	15.8%
50	50	100	85,747	82,049	83,738	37	12.2	5	12.8	55.2%	18.2%	7.5%	19.1%

For this problem a relatively small population size and small maximum number of generations (i.e., population size = 10 and max # generations = 20) provides approximately the same results with respect to student choice. Of course, these results may change depending on the underlying problem configuration, which includes the combined student choices, the weights of the choices, the project weights, and the maximum and minimum number of students required to staff each project. However, with respect to fitness value, it appears from the test data shown here that the best values are found when the population size is relatively small (i.e., 10, 20, or 30) and the maximum number of generations is moderately high (i.e., 60, 80, or 100). It is important to point out that the final fitness value includes a reward and penalty function. If a project within an individual solution has the maximum number of students assigned to it, it is rewarded and its fitness value is increased. If a project does not have the minimum number of students, it is penalized and its fitness value is decreased. Based on this convergence study, the parameters chosen for implementation of a larger problem (200 students and 80 projects), were chosen to be:

- Population (Number of Individuals) = 20 or 40
- Maximum Number of Generations = 20 - 120
- Chance = 20% or 40%

For the larger problem, the highest fitness values occur for the largest maximum number of generations tested (i.e., 120 generations) for both population sizes, 20 and 40 individuals. Interestingly, the highest fitness value of 1,484,288.8, occurred when the population size was 20 and the maximum number of generations was 120, also produced the worst results with respect to student choice. While the results were acceptable, they were lower than the other test data for the large problem. This goes to show the dual nature of the objective of trying to assign students to their choice projects while also trying to pressure the algorithm to assign students to higher weighted projects (i.e., projects with higher priority) and also give rewards when the minimum

number of students are assigned to a project and penalize when the maximum number of required students are not assigned to a project.

Table 2. Application of Converged Algorithm

				Student Choice (Average over 5 runs)					% Student Choice (Average over 5 runs)			
Population Size	Chance	Max # Generations	Fitness Value	1st	2nd	3rd	other	Total # Students	1st	2nd	3rd	other
20	20	20	1,395,539.9	141	40	10	9	200	70.5%	20.0%	5.0%	4.5%
20	20	40	1,391,030.9	143	38	10	9	200	71.5%	19.0%	5.0%	4.5%
20	20	60	1,395,624.1	143	39	10	8	200	71.5%	19.5%	5.0%	4.0%
20	20	80	1,393,201.9	143	39	10	8	200	71.5%	19.5%	5.0%	4.0%
20	20	100	1,394,532.7	143	39	9	9	200	71.5%	19.5%	4.5%	4.5%
20	20	120	1,484,288.8	129	36	11	24	200	64.5%	18.0%	5.5%	12.0%
40	40	20	1,390,652.1	143	38	10	9	200	71.5%	19.0%	5.0%	4.5%
40	40	40	1,389,698.6	143	37	9	11	200	71.5%	18.5%	4.5%	5.5%
40	40	60	1,394,664.7	143	38	11	8	200	71.5%	19.0%	5.5%	4.0%
40	40	80	1,394,043.9	143	38	11	8	200	71.5%	19.0%	5.5%	4.0%
40	40	100	1,394,043.9	141	41	10	8	200	70.5%	20.5%	5.0%	4.0%
40	40	120	1,402,551.1	144	38	11	7	200	72.0%	19.0%	5.5%	3.5%

Conclusions

A Genetic Algorithm (GA) for assigning student groups for a capstone course has been presented. The algorithm uses input of student rankings, project importance, staffing level and competencies required. This GA runs in a small fraction of the time formerly devoted to student staffing by course instructors. The particular nature of the competing parameters in this solution

(student choice versus sponsoring organization priority) are highlighted in the algorithm performance.

Appendix I

Computer Code

```
Option Explicit

Option Base 1

Public numStudents As Integer

Public numCol As Integer

Private majorCountArray() As Integer

Private majorArray() As String

Private studentMajorArray() As Integer

Public numProjects As Integer

Public students() As Variant

Public projects() As Variant

Public choiceWeights(3) As Integer

Public weightsArray() As Double

Public activeGenArray() As Integer

Public activeGenArrayFV() As Double

Public offSpringArray() As Integer

Public offSpringArrayFV() As Double

Public numOffspring As Integer

Public tempOffspring() As Integer

Public maxMajor() As Integer

Public minMajor() As Integer

Public historyArray() As Integer

Public historyArrayFV() As Double

Public crossoverProb1 As Double

Public crossoverProb2 As Double

Public mutationProb1 As Double
```

```
Public mutationProb2 As Double
Public chance As Integer
Public i As Integer
Public j As Integer
Public k As Integer
Public m As Integer
Public n As Integer
Public p As Integer
Private major As String
Private majorCount As Integer
Private diffMajorCount As Integer
Private studentMajorCount As Integer
Public maxMajors As Integer
Public minMajors As Integer
Public Count As Integer
Public Count1 As Integer
Public Count2 As Integer
Public popSize As Integer
Public rndNum As Double
Public rndNum1 As Integer
Public rndNum2 As Integer
Dim fV As Double
Public flag As Boolean
Public flag1 As Boolean
Public flag2 As Boolean
Public stop1 As Boolean
Public stop2 As Boolean
Public gen As Integer
```

```
Public maxGen As Integer
Public bestFV As Double
Public bestFVIndex As Integer
Public historyBestFV As Double
Public historyBestFVIndex As Integer
Public feasibilityArray() As Double
Public x As Double
Public y As Integer
Public z As Integer
Public t As Integer
Public start As Integer
Public finish As Integer
Public feasibilityBestIndex As Integer
Public feasibilityBestProjNum As Integer
Public feasibilityBestWeight As Integer
Public infeasibleIndex() As Integer
Public infeasibleTemp() As Integer
Public Sum As Integer
Public infeasible As Integer
Public q As Integer
Public r As Integer
Public Form As Boolean
Public sheetName As String

Sub Main()
'Student Assignment Problem
'January 4, 2011
```

```
sheetName = "Projects"

'Delete information and formating from following worksheets
ActiveWorkbook.Worksheets("matrix").Range("A1:XD1000").Clear
ActiveWorkbook.Worksheets("Output").Range("A1:XD1000").Clear
ActiveWorkbook.Worksheets("Solution Summary").Range("A1:XD1000").Clear

    'Reads input values from the "Inputs" worksheet
    With ActiveWorkbook.Worksheets("Inputs").Range("A1")
        popSize = .Offset(1, 2)
        crossoverProb1 = .Offset(2, 2)
        crossoverProb2 = .Offset(3, 2)
        mutationProb1 = .Offset(4, 2)
        mutationProb2 = .Offset(5, 2)
        chance = .Offset(6, 2)
        maxGen = .Offset(7, 2)
    End With

    'Determine the total number of students and assign each student a number in order
    Call DetermineNumStudents

    diffMajorCount = 1
    ReDim majorArray(diffMajorCount)

    'Determine the number of students for each major
    Call DetermineNumStudentsInEachMajor

    'Determine the number of projects
```

```
Call DetermineNumProjects

'Project/Major requirements from the "Project" worksheet
Call ProjectMajorMinMax

'Populate student array with each student's last name, first name, GPA, Major,
'1st, 2nd, and 3rd project choices
'NOTE: Array index corresponds to student number
Call PopStudentArray

'Populate project array with each project's name, sponsor, and weight
'NOTE: Array index corresponds to project number
Call PopProjectsArray

'Populate the choiceweights array with the weights for each of the student's
'three choices.  The weights for the three choices can be changed by changing
'the values in K4, K5, and K6 on the "Students" worksheet
Call PopChoiceWeights

'Populate the weights array.  This array contains the coefficient value
'associated with each student/project combination.  The weight for each
'combination is equal to the product for the student's GPA, the student's
'weighted choice for that project, and the project's weight
'Note: If a given project is not one of the student's three choices, it is
'given a weight of 1
Call PopWeightsArray

'Print array of weights on the "matrix" worksheet.  This worksheet shows the
```

```
'weights associated with each student/project combination.  These weights are
'the product of the project's weight, the student's GPA, and the student's
'choice weight for a given project.  If a project is not one of the student's
'three choices, it is given a choice weight of 1
Call PrintWeightsArray

'Start with generation 1
gen = 1

'Redimensionalize the activeGenArray and the activeGenArrayFV.  These arrays
'work together.  The activeGenArray contains the beginning population of
'chromosomes for each generation, and the activeGenArrayFV contains the
'corresponding fitness value for each chromosome in the activeGenArray
ReDim activeGenArray(numStudents, numProjects, popSize)
ReDim activeGenArrayFV(popSize)

'Redimensionalize the historyArray and the historyArrayFV.  These arrays
'work together.  The historyArray contains the best chromosomes found
'in each generation, and the historyArrayFV contains the corresponding fitness
'value for each chromosome in the historyArray
ReDim historyArray(numStudents, numProjects, maxGen)
ReDim historyArrayFV(maxGen)

'Creates the initial population.  The initial population is populated using the
'student's choices.  It is important that the data in the "Students" worksheet
'be sorted by Major first, and then by GPA.  Those student's with the highest
'GPA will be assigned to their choice projects, if they are not filled, first
Call InitializePop
```

```
'Determine the fitness value for each chromosome in the offspring array
Call FitnessValueInitialPop

'Run program for maximum generations.  The maximum number of generations (maxGen)
'can be changed by changing the value in cell C7 on the "Inputs" worksheet
'Generally, the larger the search space, i.e., the larger the number of possible
'Student/Project combination, the larger the maxGen should be.  This allows the
'algorithm more time to find a good solution.  However, the larger the maxGen
'the longer it will take for the algorithm to run.  It should be noted that no
'matter how long the algorithm runs, do to the randomness employeed in the algorithm,
'there is no garantee that a good solution will be reached.  If a solution does
'not appear good, the algorithm can be run again, at which point the input data
'might be changed as desired.  This algorithm will run for maxGen.
Do While gen <= maxGen

    'Copy the present generation's population of chromosomes into the offspring array
    Call CopyActivePopToOffspring

    'Crossover operation#1
    Call Crossover1

    'Crossover operation#2
    Call Crossover2

    'Mutation operation#1
    Call Mutation1
```

```
        'Mutation operation#2
        Call Mutation2

        'Determine the fitness value for each chromosome in the offspring array
        Call FitnessValue

        'Checks for feasiblity, assigns awards and penalties
        Call Feasibility

        'Select best chromosome from offSpring array and store in History Array
        Call SelectBestFromOffspring

        'Select next generations chromosomes
        Call NextGen

        gen = gen + 1
    Loop 'generational loop

    'Print the absolute best from the history array
    Call PrintAbsoluteBest

End Sub

Sub DetermineNumStudents()
    'Determine the total number of students and assign each student a number in order
    With ActiveWorkbook.Worksheets("Students").Range("A4")

        numStudents = .CurrentRegion.rows.Count - 3
```

```
        numCol = .CurrentRegion.Columns.Count


        For i = 0 To numStudents - 1
            .Offset(i, 0).Value = i + 1
        Next


    End With
End Sub


Sub DetermineNumStudentsInEachMajor()
    'Determine the number of students for each major
    With ActiveWorkbook.Worksheets("Students").Range("A3")
        For j = 0 To numCol - 1
            If .Offset(0, j).Value = "Major" Then
                major = .Offset(1, j).Value
                studentMajorCount = 0
                majorCount = 0
                diffMajorCount = 1



                For i = 1 To numStudents
                    If major = .Offset(i, j).Value Then
                        majorCount = majorCount + 1
                        studentMajorCount = studentMajorCount + 1
                    Else
                        ReDim Preserve majorArray(diffMajorCount)
                        majorArray(diffMajorCount) = major 'records name of major
                        ReDim Preserve majorCountArray(diffMajorCount)
```

```
                majorCountArray(diffMajorCount) = majorCount 'records the number students in each array
                ReDim Preserve studentMajorArray(diffMajorCount)
                studentMajorArray(diffMajorCount) = studentMajorCount 'records the number of students in each
array; Redundant
                majorCount = 1
                studentMajorCount = 1
                diffMajorCount = diffMajorCount + 1
                major = .Offset(i, j).Value
            End If
        Next
            ReDim Preserve majorArray(diffMajorCount)
            majorArray(diffMajorCount) = major
            ReDim Preserve majorCountArray(diffMajorCount)
            majorCountArray(diffMajorCount) = majorCount
            ReDim Preserve studentMajorArray(diffMajorCount)
            studentMajorArray(diffMajorCount) = studentMajorCount
          Exit For
      End If
    Next
  End With
End Sub

Sub DetermineNumProjects()
  'Determine the number of projects
  With ActiveWorkbook.Worksheets(sheetName).Range("A4")

    numProjects = .CurrentRegion.rows.Count - 3
    numCol = .CurrentRegion.Columns.Count
```

```
        For i = 0 To numProjects - 1
            .Offset(i, 0).Value = i + 1
        Next
    End With
End Sub

Sub ProjectMajorMinMax()
    'Project/Major requirements from the "Project" worksheet
    'Records the maximum number of students from each major for each
    'project, and the minimum number of students from each major
    'for each project.  This information is used in the feasibility
    'sub-routine to reward solutions with project that have at least
    'the minumum number of each major and penalize those projects that
    'have more than the maximum number of each major.  These are soft
    'contraints however and do not gaurantee that the final, best,
    'solution will not violate the min/max for each project
    ReDim maxMajor(numProjects, diffMajorCount)
    ReDim minMajor(numProjects, diffMajorCount)
    Count = 0

    With ActiveWorkbook.Worksheets(sheetName).Range("A3")

        'Determine the maximum number of each major required for each project in the Project worksheet
        Count = 4
        Do While Count < diffMajorCount + 4
            For j = 1 To diffMajorCount
                For i = 1 To numProjects
```

```
                maxMajor(i, j) = .Offset(i, Count).Value
            Next
            Count = Count + 1
        Next
    Loop

    'Determine the minimum number of each major required for each project in the Project worksheet
    Count = 7
    Do While Count < diffMajorCount + 7

        For j = 1 To diffMajorCount
            For i = 1 To numProjects
                minMajor(i, j) = .Offset(i, Count).Value
            Next
            Count = Count + 1
        Next
    Loop
  End With
End Sub

Sub PopStudentArray()
    'Populate student array with each student's last name, first name, GPA, Major,
    '1st, 2nd, and 3rd project choices
    'NOTE: Array index corresponds to student number
    ReDim students(numStudents, 7)

    With ActiveWorkbook.Worksheets("Students").Range("A3")
```

```
        For i = 1 To numStudents
            students(i, 1) = .Offset(i, 1).Value 'last name
            students(i, 2) = .Offset(i, 2).Value 'first name
            students(i, 3) = .Offset(i, 3).Value 'GPA
            students(i, 4) = .Offset(i, 4).Value 'Major
            students(i, 5) = .Offset(i, 5).Value '1st choice
            students(i, 6) = .Offset(i, 6).Value '2nd choice
            students(i, 7) = .Offset(i, 7).Value '3rd choice
        Next
    End With
End Sub

Sub PopProjectsArray()
    'Populate project array with each project's name, sponsor, and weight
    'NOTE: Array index corresponds to project number
    ReDim projects(numProjects, 3)

    With ActiveWorkbook.Worksheets(sheetName).Range("A3")
        For i = 1 To numProjects
            projects(i, 1) = .Offset(i, 1).Value 'name
            projects(i, 2) = .Offset(i, 2).Value 'sponsor
            projects(i, 3) = .Offset(i, 3).Value 'weight
        Next
    End With
End Sub

Sub PopChoiceWeights()
    'Populate the choiceweights array with the weights for each of the student's
```

```
    'three choices.  The weights for the three choices can be changed by changing
    'the values in K4, K5, and K6 on the "Students" worksheet
    With ActiveWorkbook.Worksheets("Students").Range("J3")
        For i = 1 To 3
            choiceWeights(i) = .Offset(i, 1).Value '1st choice weight
            choiceWeights(i) = .Offset(i, 1).Value '2nd choice weight
            choiceWeights(i) = .Offset(i, 1).Value '3rd choice weight
        Next
    End With
End Sub

Sub PopWeightsArray()
    'Populate the weights array.  This array contains the coefficient value
    'associated with each student/project combination.  The weight for each
    'combination is equal to the product for the student's GPA, the student's
    'weighted choice for that project, and the project's weight
    'Note: If a given project is not one of the student's three choices, it is
    'given a weight of 1
    ReDim weightsArray(numStudents, numProjects)

    For i = 1 To numStudents
        For j = 1 To numProjects
            If students(i, 5) = j Then '1st choice
                weightsArray(i, j) = choiceWeights(1) * projects(j, 3) * students(i, 3)
            ElseIf students(i, 6) = j Then '2nd choice
                weightsArray(i, j) = choiceWeights(2) * projects(j, 3) * students(i, 3)
            ElseIf students(i, 7) = j Then '3rd choice
                weightsArray(i, j) = choiceWeights(3) * projects(j, 3) * students(i, 3)
```

```
        Else    'Project is not one of the students 3 choices, then weight = 1
            weightsArray(i, j) = projects(j, 3) * students(i, 3)
        End If
      Next
   Next
End Sub

Sub PrintWeightsArray()
   'Print array of weights on the "matrix" worksheet.  This worksheet shows the
   'weights associated with each student/project combination.  These weights are
   'the product of the project's weight, the student's GPA, and the student's
   'choice weight for a given project.  If a project is not one of the student's
   'three choices, it is given a choice weight of 1
   ActiveWorkbook.Worksheets("matrix").Range("A1:XD1000").Clear

   With ActiveWorkbook.Worksheets("matrix").Range("A1")
      For i = 1 To numStudents
         For j = 1 To numProjects
            .Offset(i, j).Value = weightsArray(i, j)

            If students(i, 5) = j Then '1st choice
               .Offset(i, j).Interior.Color = vbBlue
            ElseIf students(i, 6) = j Then '2nd choice
               .Offset(i, j).Interior.Color = vbRed
            ElseIf students(i, 7) = j Then '3rd choice
               .Offset(i, j).Interior.Color = vbGreen
            End If
         Next j
```

```
        Next i
    End With
End Sub

Sub InitializePop()
    'Creates the initial population.  The initial population is populated using the
    'student's choices.  It is important that the data in the "Students" worksheet
    'be sorted by Major first, and then by GPA.  Those student's with the highest
    'GPA will be assigned to their choice projects, if they are not filled, first
    Dim k2 As Integer
    Dim j2 As Integer
    Dim start2 As Integer
    Dim finish2 As Integer
    Dim Count2B As Integer
    Dim Count1B As Integer
    Dim CountB As Integer
    Dim m2 As Integer
    Dim i2 As Integer
    Dim projFill() As Integer
    Dim not1st As Boolean
    Dim not2nd As Boolean
    Dim not3rd As Boolean
    Dim tempArray() As Integer
    Dim temp As Integer
    Dim tempCount As Integer
    Dim turn As Integer
    Dim sort1 As Integer
    Dim sort2 As Integer
```

```
ReDim tempArray(numProjects - 3)
ReDim projFill(numProjects, diffMajorCount)
Dim choiceArray(3) As Integer
For j2 = 1 To popSize
    flag = False
    flag1 = True
    not1st = False
    not2nd = False
    not3rd = False
    tempCount = 1
      start2 = 1
      finish2 = 0
      Count2B = 0
      Count1B = 0
      CountB = 0

  For m2 = 1 To diffMajorCount
          start2 = finish2 + 1
          CountB = majorCountArray(m2) 'How many students are in each major
          finish2 = finish2 + CountB

     For i2 = start2 To finish2
        'Clears out tempArray
        For p = 1 To numProjects - 3
           tempArray(p) = 0
         Next
```

```
turn = 0
Do Until turn >= 3
turn = turn + 1

   flag = False
   flag1 = True

   'Assigns student to their 1st choice if the max number of majors
   'for that project has not been met
   If turn = 1 Then
      For k2 = 1 To numProjects
         If students(i2, 5) = k2 Then
            If projFill(k2, m2) < maxMajor(k2, m2) Then
               projFill(k2, m2) = projFill(k2, m2) + 1
               activeGenArray(i2, k2, j2) = 1
               flag = True
               turn = 4
               k2 = numProjects
            End If
         End If
      Next k2
   End If

   'Assigns student to their 2nd choice if the max number of majors
   'for that project has not been met
   If turn = 2 Then
      For k2 = 1 To numProjects
         If students(i2, 6) = k2 Then
```

```
            If projFill(k2, m2) < maxMajor(k2, m2) Then
                projFill(k2, m2) = projFill(k2, m2) + 1
                activeGenArray(i2, k2, j2) = 1
                flag = True
                'flag1 = False
                'not2nd = True
                turn = 4
                k2 = numProjects
            End If
        End If
    Next k2
End If

'Assigns student to their 3rd choice if the max number of majors
'for that project has not been met
If turn = 3 Then
    For k2 = 1 To numProjects
        If students(i2, 7) = k2 Then
            If projFill(k2, m2) < maxMajor(k2, m2) Then
                projFill(k2, m2) = projFill(k2, m2) + 1
                activeGenArray(i2, k2, j2) = 1
                flag = True
                'flag1 = False
                'not3rd = True
                turn = 4
                k2 = numProjects
            End If
        End If
```

```
        Next k2
    End If

    If turn = 4 Then
        choiceArray(1) = students(i2, 5)
        choiceArray(2) = students(i2, 6)
        choiceArray(3) = students(i2, 7)

        For j = 1 To 3
            For i = 1 To 2
                If choiceArray(i) > choiceArray(i + 1) Then
                    sort1 = choiceArray(i)
                    choiceArray(i) = choiceArray(i + 1)
                    choiceArray(i + 1) = sort1
                End If
            Next
        Next
    End If
Loop

'If all three student choices are not available, then
'the remaining projects are listed in a temporary array, tempArray
    If flag = False Then

    tempCount = 1
    For k2 = 1 To numProjects
        'tempCount = 1
```

```
    If tempCount <= numProjects - 3 Then
        If choiceArray(1) = k2 Then
        ElseIf choiceArray(2) = k2 Then
        ElseIf choiceArray(3) = k2 Then
        Else
            tempArray(tempCount) = k2
            tempCount = tempCount + 1
        End If
    End If
 Next k2

'If a student has not been assigned to a project yet
q = numProjects - 3
flag1 = False
flag = True
Do Until flag1 = True
    rndNum = WorksheetFunction.RandBetween(1, q)
        temp = tempArray(rndNum)
        If projFill(temp, m2) < maxMajor(temp, m2) Then
            projFill(temp, m2) = projFill(temp, m2) + 1
            activeGenArray(i2, temp, j2) = 1
            flag = True
            flag1 = True
        End If
        For t = rndNum To numProjects - 4
           tempArray(rndNum) = tempArray(rndNum + 1)
           rndNum = rndNum + 1
```

```
                    Next
                    If q > 1 Then
                        q = q - 1
                    Else
                        flag1 = True
                    End If
                Loop
            End If
        Next i2
    Next m2
Next j2
End Sub

Sub CopyActivePopToOffspring()
    'Copy the present generation's population of chromosomes into the offspring array
    numOffspring = popSize
    ReDim offSpringArray(numStudents, numProjects, numOffspring)

    For j = 1 To popSize
        For i = 1 To numStudents
            For k = 1 To numProjects
                offSpringArray(i, k, j) = activeGenArray(i, k, j)
            Next
        Next
    Next
End Sub

Sub Crossover1()
```

```
Count = 0

Do Until Count = chance
    Count = Count + 1
    Randomize
    rndNum = Rnd()

    'If a randomly generated number is less than or equal to the probability of crossover
    '(crossoverProb), then perform crossover
    If rndNum <= crossoverProb1 Then
        ReDim tempOffspring(numStudents, numProjects, 4)

        'Randomly select two Parent chromosomes from the population
        For m = 1 To 2
            rndNum1 = WorksheetFunction.RandBetween(1, popSize)
                For j = 1 To numStudents
                    For i = 1 To numProjects
                        tempOffspring(j, i, m) = activeGenArray(j, i, rndNum1)
                    Next i
                Next j
        Next m

        'Randomly select a crossover point
        rndNum2 = WorksheetFunction.RandBetween(1, numStudents)

        'Preform crossover at crossover point on two Parent chromosomes
        For m = 1 To 2
            For i = 1 To numProjects
```

```
            For j = 1 To rndNum2
               tempOffspring(j, i, 3) = tempOffspring(j, i, 1)
               tempOffspring(j, i, 4) = tempOffspring(j, i, 2)
            Next j
            For j = rndNum2 + 1 To numStudents
               tempOffspring(j, i, 3) = tempOffspring(j, i, 2)
               tempOffspring(j, i, 4) = tempOffspring(j, i, 1)
            Next j
         Next i
      Next m

      'Increase the number of offspring by two and increase size of
      'offSpringArray by 2
      numOffspring = numOffspring + 2
      ReDim Preserve offSpringArray(numStudents, numProjects, numOffspring)

      'Copy two new offspring into the offSpringArray
      For j = 1 To numStudents
         For i = 1 To numProjects
            offSpringArray(j, i, numOffspring - 1) = tempOffspring(j, i, 3)
            offSpringArray(j, i, numOffspring) = tempOffspring(j, i, 4)
         Next i
      Next j
   End If
Loop
End Sub

Sub Crossover2()
```

```
Count = 0

Do Until Count = chance
    Count = Count + 1
    Randomize
    rndNum = Rnd()

    'If a randomly generated number is less than or equal to the probability of crossover
    '(crossoverProb), then perform crossover
    If rndNum <= crossoverProb2 Then
        ReDim tempOffspring(numStudents, numProjects, 4)

        'Randomly select two Parent chromosomes from the population
        For m = 1 To 2
            rndNum1 = WorksheetFunction.RandBetween(1, popSize)
                For j = 1 To numStudents
                    For i = 1 To numProjects
                        tempOffspring(j, i, m) = activeGenArray(j, i, rndNum1)
                    Next i
                Next j
        Next m

        'Randomly select a crossover point
        rndNum2 = WorksheetFunction.RandBetween(1, numProjects)

        'Preform crossover at crossover point on two Parent chromosomes
        For m = 1 To 2
            For i = 1 To numStudents
```

```
            For j = 1 To rndNum2
               tempOffspring(i, j, 3) = tempOffspring(i, j, 1)
               tempOffspring(i, j, 4) = tempOffspring(i, j, 2)
            Next j
            For j = rndNum2 + 1 To numProjects
               tempOffspring(i, j, 3) = tempOffspring(i, j, 2)
               tempOffspring(i, j, 4) = tempOffspring(i, j, 1)
            Next j
         Next i
      Next m

      'Increase the number of offspring by two and increase size of
      'offSpringArray by 2
      numOffspring = numOffspring + 2
      ReDim Preserve offSpringArray(numStudents, numProjects, numOffspring)

      'Copy two new offspring into the offSpringArray
      For j = 1 To numStudents
         For i = 1 To numProjects
            offSpringArray(j, i, numOffspring - 1) = tempOffspring(j, i, 3)
            offSpringArray(j, i, numOffspring) = tempOffspring(j, i, 4)
         Next i
      Next j
   End If
 Loop
End Sub

Sub Mutation1()
```

```
Count = 0
Do While Count <> chance
   Count = Count + 1
   Randomize
   rndNum = Rnd()

   'If a randomly generated number is less than or equal to the probability of
   'mutation operator #1(mutationProb1), then perform mutation operator #1
   If rndNum <= mutationProb1 Then
   ReDim tempOffspring(numStudents, numProjects, 3)

      'Randomly select one Parent chromosome from the population
      rndNum1 = WorksheetFunction.RandBetween(1, popSize)
      For j = 1 To numStudents
         For i = 1 To numProjects
            tempOffspring(j, i, 1) = activeGenArray(j, i, rndNum1)
            tempOffspring(j, i, 2) = activeGenArray(j, i, rndNum1)
         Next
      Next

      'Randomly select two students from Parent chromosomes
      rndNum1 = WorksheetFunction.RandBetween(1, numStudents)
      rndNum2 = WorksheetFunction.RandBetween(1, numStudents)

      'Randomly swap student assignments
      For i = 1 To numProjects
        tempOffspring(rndNum1, i, 2) = tempOffspring(rndNum2, i, 1)
      Next i
```

```
        For i = 1 To numProjects
          tempOffspring(rndNum2, i, 2) = tempOffspring(rndNum1, i, 1)
        Next i

        'Increase the number of offspring by two and increase size of
        'offSpringArray by 1
        numOffspring = numOffspring + 1
        ReDim Preserve offSpringArray(numStudents, numProjects, numOffspring)

        'Copy two new offspring into the offSpringArray
        For j = 1 To numStudents
          For i = 1 To numProjects
            offSpringArray(j, i, numOffspring) = tempOffspring(j, i, 2)
          Next
        Next
      End If
    Loop
End Sub

Sub Mutation2()
    Count = 0
    Do While Count <> chance
      Count = Count + 1
      Randomize
      rndNum = Rnd()

      'If a randomly generated number is less than or equal to the probability of
```

```
'mutation operator #2(mutationProb2), then perform mutation operator #2
If rndNum <= mutationProb2 Then
ReDim tempOffspring(numStudents, numProjects, 3)

   'Randomly select one Parent chromosome from the population
   rndNum1 = WorksheetFunction.RandBetween(1, popSize)
   For j = 1 To numStudents
      For i = 1 To numProjects
         tempOffspring(j, i, 1) = activeGenArray(j, i, rndNum1)
      Next
   Next

   'Randomly select a student and a project from Parent chromosomes
   rndNum1 = WorksheetFunction.RandBetween(1, numStudents)
   rndNum2 = WorksheetFunction.RandBetween(1, numProjects)

      For i = 1 To numProjects
         If tempOffspring(rndNum1, i, 1) = 1 Then
            q = i
         End If
      Next i

      'Swap the assigned project for a student with a randomly selected project
      tempOffspring(rndNum1, q, 1) = 0
      tempOffspring(rndNum1, rndNum2, 1) = 1

      'Increase the number of offspring by two and increase size of
      'offSpringArray by 1
```

```
            numOffspring = numOffspring + 1
            ReDim Preserve offSpringArray(numStudents, numProjects, numOffspring)

            'Copy two new offspring into the offSpringArray
            For j = 1 To numStudents
               For i = 1 To numProjects
                  offSpringArray(j, i, numOffspring) = tempOffspring(j, i, 1)
               Next i
            Next j
         End If
      Loop
End Sub
Sub FitnessValueInitialPop()
   'Determine the fitness value for each chromosome in the offspring array
  ' ReDim offSpringArrayFV(numOffspring)

   For j = 1 To popSize
      fV = 0
      For i = 1 To numStudents
         For k = 1 To numProjects
            fV = fV + activeGenArray(i, k, j) * weightsArray(i, k)
         Next
      Next
      activeGenArrayFV(j) = fV
   Next
End Sub
Sub FitnessValue()
   'Determine the fitness value for each chromosome in the offspring array
```

```
    ReDim offSpringArrayFV(numOffspring)


    For j = 1 To numOffspring
        fV = 0
        For i = 1 To numStudents
            For k = 1 To numProjects
                fV = fV + offSpringArray(i, k, j) * weightsArray(i, k)
            Next
        Next
        offSpringArrayFV(j) = fV
    Next
End Sub


Sub Feasibility()
    'Checks for feasiblity, assigns awards and penalties
    Dim i1 As Integer
    Dim j1 As Integer
    Dim k1 As Integer
    Dim m1 As Integer
    Dim p1 As Integer
    Dim t1 As Integer
    Dim CountA As Integer
    Dim Count1A As Integer
    Dim Count2A As Integer
    Dim start1 As Integer
    Dim finish1 As Integer
    Dim infeasIndex() As Integer
    Dim total As Integer
```

```
total = 0
For k = 1 To numOffspring
   For i = 1 To numStudents
      For j = 1 To numProjects
         'For student, check to make sure that they have not been
         'assigned to more than one project
         If offSpringArray(i, j, k) = 1 Then
            ReDim Preserve infeasIndex(total + 1)
               infeasIndex(total + 1) = j
               total = total + 1
         End If
      Next
         'If a student is assigned to more than one project, then
         'randomly select a project for which the student is assigned
         'and remove that assignment.  Continue this process until
         'the student is only assigned to one project
         If total > 1 Then
            Do Until total = 1
               rndNum = WorksheetFunction.RandBetween(1, total)
               offSpringArray(i, infeasIndex(rndNum), k) = 0
               total = total - 1
            Loop
         End If
         'If a student is not assigned to any project, then randomly select a project
         If total = 0 Then
            rndNum = WorksheetFunction.RandBetween(1, numProjects)
            offSpringArray(i, rndNum, k) = 1
```

```
            End If
            total = 0
        Next
    Next

    For j1 = 1 To numOffspring
        For k1 = 1 To numProjects
            start1 = 1
            finish1 = 0
            Count2A = 0
            CountA = 0

            If k1 <> numProjects Then
                For m1 = 1 To diffMajorCount
                    start1 = finish1 + 1
                    CountA = majorCountArray(m1) 'How many students are in each major
                    finish1 = finish1 + CountA
                    For i1 = start1 To finish1
                        If offSpringArray(i1, k1, j1) = 1 Then
                            Count2A = Count2A + 1
                        End If
                    Next i1

                    For p1 = k1 + 1 To numProjects
                        For t1 = start1 To finish1
                            If offSpringArray(t1, k1, j1) = 1 Then
                                If t1 = start1 Then
                                    feasibilityBestIndex = start1              'student number
```

```
                    feasibilityBestProjNum = p1                'project number
                    feasibilityBestWeight = weightsArray(t1, p1) 'weight
                  Else
                    y = t1                  'student number
                    z = p1                  'project number
                    x = weightsArray(t1, p1)  'weight
                  End If

                  If x > feasibilityBestWeight Then
                    feasibilityBestWeight = x
                    feasibilityBestIndex = y
                    feasibilityBestProjNum = z
                  End If
                End If
              Next t1
            Next p1
          Next m1
        End If
      Next k1
    Next j1

'***Reward
    Dim k2 As Integer
    Dim j2 As Integer
    Dim start2 As Integer
    Dim finish2 As Integer
    Dim Count2B As Integer
    Dim Count1B As Integer
```

```
Dim CountB As Integer
Dim m2 As Integer
Dim i2 As Integer

For j2 = 1 To numOffspring
   For k2 = 1 To numProjects
      start2 = 1
      finish2 = 0
      Count2B = 0
      Count1B = 0
      CountB = 0
         For m2 = 1 To diffMajorCount
            start2 = finish2 + 1
            CountB = majorCountArray(m2) 'How many students are in each major
            finish2 = finish2 + CountB
            For i2 = start2 To finish2
               If offSpringArray(i2, k2, j2) = 1 Then
                  Count2B = Count2B + 1
               End If
            Next i2

            'For each project, if at least the minimum number of a major has been assigned,
            'then reward chromosome
            If Count2B >= minMajor(k2, m2) Then
               offSpringArrayFV(j2) = offSpringArrayFV(j2) + projects(k2, 3) * 1 '<--2
            End If
         Next m2
   Next k2
```

```
    Next j2


'***Penalty
    Dim j3 As Integer
    Dim k3 As Integer
    Dim start3 As Integer
    Dim finish3 As Integer
    Dim Count2C As Integer
    Dim CountC As Integer
    Dim m3 As Integer
    Dim i3 As Integer


      For j3 = 1 To numOffspring
        For k3 = 1 To numProjects
          start3 = 1
          finish3 = 0
          Count2C = 0
          CountC = 0
            For m3 = 1 To diffMajorCount
            start3 = finish3 + 1
            CountC = majorCountArray(m3) 'How many students are in each major
            finish3 = finish3 + CountC
              For i3 = start3 To finish3
                If offSpringArray(i3, k3, j3) = 1 Then
                  Count2C = Count2C + 1
                End If
              Next i3
```

```
            'For each project, if the more than the maximum number of a major
            'has been assigned, then penalize chromosome
            If Count2C > maxMajor(k3, m3) Then
               If offSpringArrayFV(j3) >= projects(k3, 3) Then
                  offSpringArrayFV(j3) = offSpringArrayFV(j3) - projects(k3, 3) * 1 '<--3
               Else
                  offSpringArrayFV(j3) = 0
               End If
            End If
         Next m3
      Next k3
   Next j3
End Sub

Sub SelectBestFromOffspring()
   'Select best chromosome from offSpring array and store in History Array
   bestFV = offSpringArrayFV(1)
   bestFVIndex = 1
   For m = 2 To numOffspring
      If bestFV < offSpringArrayFV(m) Then
         bestFV = offSpringArrayFV(m)
         bestFVIndex = m
      End If
   Next m

   For i = 1 To numStudents
      For j = 1 To numProjects
         historyArray(i, j, gen) = offSpringArray(i, j, bestFVIndex)
```

```
        historyArrayFV(gen) = offSpringArrayFV(bestFVIndex)


        If gen = 1 Then
            historyBestFV = historyArrayFV(gen)
            historyBestFVIndex = gen
        ElseIf historyArrayFV(gen) > historyBestFV Then
            historyBestFV = historyArrayFV(gen)
            historyBestFVIndex = gen
        End If
      Next j
   Next i
End Sub


Sub NextGen()
   'Select next generations chromosomes
   For k = 1 To popSize
   rndNum1 = WorksheetFunction.RandBetween(1, numOffspring)
      For n = 1 To 2
       rndNum2 = WorksheetFunction.RandBetween(1, numOffspring)
        For j = 1 To numProjects
            For i = 1 To numStudents
                If offSpringArrayFV(rndNum1) > offSpringArrayFV(rndNum2) Then
                    activeGenArray(i, j, popSize) = offSpringArray(i, j, rndNum1)
                Else
                    activeGenArray(i, j, popSize) = offSpringArray(i, j, rndNum2)
                End If
            Next i
        Next j
```

```
            rndNum1 = rndNum2
        Next n
    Next k
End Sub

Sub PrintAbsoluteBest()
    Dim number As Integer
    Dim Addup As Integer

    ActiveWorkbook.Worksheets("Output").Range("A1:XD1000").Clear

    With ActiveWorkbook.Worksheets("Output").Range("C5")
     Count = 0

        With .Offset(-1, Count - 1)
            .Value = "Fitness Value = "
            .Font.Bold = True
            .Columns.AutoFit
        End With
        With .Offset(-1, 0)
            .Value = historyBestFV
            .Font.Bold = True
            .Columns.AutoFit
        End With
        With .Offset(0, Count)
            .Value = "Last Name"
            .Font.Bold = True
            .Columns.AutoFit
```

```
End With
With .Offset(0, Count + 1)
    .Value = "First Name"
    .Font.Bold = True
    .Columns.AutoFit
End With
With .Offset(0, Count + 2)
    .Value = "GPA"
    .Font.Bold = True
    .Columns.AutoFit
End With
With .Offset(0, Count + 3)
    .Value = "Major"
    .Font.Bold = True
    .Columns.AutoFit
End With

For i = 1 To numStudents
    With .Offset(i, Count)
        .Value = students(i, 1)    'last name
        .Columns.AutoFit
    End With
    With .Offset(i, Count + 1)
        .Value = students(i, 2)    'first name
        .Columns.AutoFit
    End With
    With .Offset(i, Count + 2)
        .Value = students(i, 3)    'GPA
```

```
    .NumberFormat = "0.0"
    Columns.AutoFit
End With
With .Offset(i, Count + 3)
    .Value = students(i, 4)    'Major
    .Columns.AutoFit
End With
Count = Count + 3
For j = 1 To numProjects
    With .Offset(0, j + 3)
        .Value = "P" & j
        .Font.Bold = True
    End With
    With .Offset(i, Count + j)
        .Value = historyArray(i, j, historyBestFVIndex)
        If historyArray(i, j, historyBestFVIndex) = 1 Then
            .Font.Bold = True
        End If
    End With

    If students(i, 5) = j Then '1st choice
        With .Offset(i, Count + j).Interior
            .Pattern = xlSolid
            .PatternColorIndex = xlAutomatic
            .ThemeColor = xlThemeColorAccent1
            .TintAndShade = 0.399975585192419
            .PatternTintAndShade = 0
        End With
```

```
            ElseIf students(i, 6) = j Then '2nd choice
                With .Offset(i, Count + j).Interior
                    .Pattern = xlSolid
                    .PatternColorIndex = xlAutomatic
                    .ThemeColor = xlThemeColorAccent2
                    .TintAndShade = 0.399975585192419
                    .PatternTintAndShade = 0
                End With
            ElseIf students(i, 7) = j Then '3rd choice
                With .Offset(i, Count + j).Interior
                    .Pattern = xlSolid
                    .PatternColorIndex = xlAutomatic
                    .ThemeColor = xlThemeColorAccent3
                    .TintAndShade = 0.399975585192419
                    .PatternTintAndShade = 0
                End With
            End If
        Next j
        Count = 0
    Next i
End With

With ActiveWorkbook.Worksheets("Output").Range("B4")
    With .Offset(1, 6 + numProjects)
        .Value = "Check"
        .Font.Bold = True
    End With
    For i = 1 To numStudents
```

```
        Addup = 0
        For j = 1 To numProjects
            Addup = Addup + .Offset(i + 1, j + 4)
            If j = numProjects Then
                .Offset(i + 1, numProjects + 6).Value = Addup
            End If
        Next j
    Next i
End With

With ActiveWorkbook.Worksheets("Output").Range("B4")

With .Offset(numStudents + 3, 4)
    .Value = "Sum = "
    .HorizontalAlignment = xlRight
    .Font.Bold = True
End With
     For i = 1 To numProjects
        Addup = 0
        For j = 1 To numStudents
            Addup = Addup + .Offset(j + 1, i + 4)
            If j = numStudents Then
                .Offset(numStudents + 3, i + 4).Value = Addup
            End If
        Next j
    Next i
End With
```

```
    With ActiveWorkbook.Worksheets("Output").Range("B4")
      Range(.Offset(0, 0), .Offset(10000, 1000)).Columns.AutoFit
    End With

End Sub

Sub SolutionSummary()

    Dim number As Integer
    Dim number2 As Integer

    ActiveWorkbook.Worksheets("Solution Summary").Range("A1:XD1000").Clear

    With ActiveWorkbook.Worksheets("Solution Summary").Range("A1")
      With .Offset(0, 0)
        .Value = "Solution Summary"
        .Font.Size = 14
        .Font.Bold = True
      End With

        Count = 0
        number = 0
        number2 = 0
        For j = 1 To numProjects
        With .Offset(j + number + 1, Count)
          .Value = "Project Number: " & j
          .Font.Bold = True
        End With
```

```
With .Offset(j + number + 2, Count)
    .Value = "Project Name: " & projects(j, 1)  'name
    .Font.Bold = True
End With
With .Offset(j + number + 3, Count)
    .Value = "Project Sponsor: " & projects(j, 2)   'sponsor
    .Font.Bold = True
End With
With .Offset(j + number + 4, Count)
    .Value = "Project Weight: " & projects(j, 3) 'weight
    .Font.Bold = True
End With

With .Offset(j + number + 6, Count + 1)
    .Value = "Last Name"
    .Font.Bold = True
End With
With .Offset(j + number + 6, Count + 2)
    .Value = "First Name"
    .Font.Bold = True
End With
With .Offset(j + number + 6, Count + 3)
    .Value = "GPA"
    .Font.Bold = True
End With
With .Offset(j + number + 6, Count + 4)
    .Value = "Major"
    .Font.Bold = True
```

```
End With
With .Offset(j + number + 6, Count + 5)
   .Value = "Choice?"
   .Font.Bold = True
End With


  number = number + 6


   For i = 1 To numStudents
      If historyArray(i, j, historyBestFVIndex) = 1 Then
         number2 = number2 + 1
         With .Offset(j + number + number2, Count + 1)
            .Value = students(i, 1) 'last name
            .Columns.AutoFit
         End With
         With .Offset(j + number + number2, Count + 2)
            .Value = students(i, 2)    'first name
            .Columns.AutoFit
         End With
         With .Offset(j + number + number2, Count + 3)
            .Value = students(i, 3)    'GPA
            .NumberFormat = "0.0"
            Columns.AutoFit
         End With
         With .Offset(j + number + number2, Count + 4)
            .Value = students(i, 4)    'Major
            .Columns.AutoFit
         End With
```

```
        If students(i, 5) = j Then '1st choice
            With .Offset(j + number + number2, Count + 5)
                .Value = "1st"
                .Columns.AutoFit
            End With

        ElseIf students(i, 6) = j Then '2nd choice
            With .Offset(j + number + number2, Count + 5)
                .Value = "2nd"
                .Columns.AutoFit
            End With

        ElseIf students(i, 7) = j Then '3rd choice
            With .Offset(j + number + number2, Count + 5)
                .Value = "3rd"
                .Columns.AutoFit
            End With
        Else
            With .Offset(j + number + number2, Count + 5)
                .Value = "--"
                .Columns.AutoFit
            End With
        End If
    End If
Next i
Count = 0
number = number + number2
```

```
            number2 = 0

        Next j

    End With


    With ActiveWorkbook.Worksheets("Solution Summary").Range("A1")

        Range(.Offset(0, 0), .Offset(1000, 25)).Columns.AutoFit

    End With


End Sub
```

---

1. Aytug , Khouja, et al. "Use of genetic algorithms to solve production and operations management problems: a review." *International Journal of Production Research* **41** (17) pg 3955 – 4009, 2003

2. Michalewicz, *Genetic algorithms + data structures = evolution programs*, New York: Springer, 1992

3. Vergara, Khouja, and Michalewicz, "An evolutionary algorithm for optimizing material flow in supply chains," *Computers and Industrial Engineering* 43 (3) pg 407- 421, 2002