INFORMS Transactions on Education

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Optimizing the Assignment of Students to Classes in an Elementary School

Binyamin Krauss, Jon Lee, Daniel Newman

To cite this article:
Binyamin Krauss, Jon Lee, Daniel Newman (2013) Optimizing the Assignment of Students to Classes in an Elementary School. INFORMS Transactions on Education 14(1):39-44. http://dx.doi.org/10.1287/ited.2013.0111

Vol. 14, No. 1, September 2013, pp. 39–44
ISSN 1532-0545 (online)
INFORMS Transactions on Education
http://dx.doi.org/10.1287/ited.2013.0111

Optimizing the Assignment of Students to Classes in an Elementary School

Binyamin Krauss
Rabbi, Principal, SAR Academy, New York, New York 10471,
kraussb@saracademy.org

Jon Lee
Industrial and Operations Engineering Department, University of Michigan, Ann Arbor, Michigan 48109,
jonxlee@umich.edu

Daniel Newman
New York, New York 10025,
dtnewman@gmail.com

Every summer, the Salanter Akiba Riverdale (SAR) Academy must create class placements for their elementary school students. Each grade of 80 to 100 students must be divided into four classes. In assigning students to classes, the school administration aims to foster a positive social and educational environment for students while satisfying placement requests and recommendations from parents, teachers, and school therapists. The school must satisfy several constraints such as not placing certain pairs of students in the same class or keeping boy/girl ratios balanced. The process of creating optimal class placements by hand can be laborious and difficult, especially for grades with many constraints to satisfy. This paper describes a model that is being used to assist SAR Academy with creating class placements. Following the constraints and objectives given by administrators at the school, we describe an integer-programming model for satisfying placement constraints and heuristics to further improve on the outputs of the integer-programming model. The results of this process were successfully used to assist administrators in assigning students in one grade for the 2012–13 school year, and SAR Academy plans to use the model for help with future class placements.

Key words: assignment problem; class assignment; integer programming; genetic algorithm
History: Received: September 2012; accepted: March 2013.

1. Introduction

Salanter Akiba Riverdale (SAR) Academy is a coeducational, private Modern Orthodox Jewish day school located in the Riverdale section of the Bronx, New York City. Every summer, SAR Academy must assign students in their elementary school to classes. Generally, each grade of 80 to 100 students is split into four different classes. The school's administration aims to create classes that foster a positive social and educational environment for students and that make teaching easier for teachers. Typically, a few different parties provide input that administrators use to assign students to classes. Parents may provide information such as which other students their children should be with for social reasons. Teachers and school therapists also provide input on how to best meet each student's social and educational needs. Assigning students to classes is a difficult task. Recommendations and requests from different parties often conflict with each other and it falls on the administration to prioritize which requests should be fulfilled over others. For example, the administration might choose to separate two students with behavioral issues even if their parents request that their children be placed together. Poor student placement may make teaching difficult and may result in dissatisfaction from students and parents. Administrators gave the following general criteria for how they assign students into classes:

- The school guarantees that each student will be placed with at least one child from a list of up to four children provided by that student. For situations in which this is not possible, they are in touch with the parents to explain and discuss. Teachers and therapists also frequently recommend students who should be placed together and administrators try to take their recommendations into account.
- Parents, teachers, and therapists may request that certain students not be placed together for social,

behavioral, or other reasons. Additionally, according to school policy, twins may not be placed into the same class.

- One or two of the classes are designated as "inclusion classes." Certain students with an academic need for more closely supervised study must be placed in such classes (along with other students not in such a category). These classes are staffed with teachers having skills and experience appropriate for such students.
- The four classes should all have roughly the same number of students, although the inclusion class(es) may have a few more if necessary in generating a feasible solution.
- Students in the school come from various locations in the New York City area. It is preferable for students to have at least one or two other children from their neighborhood in their class, but it is also preferable that not too many students from one neighborhood are in the same class.
- The school prefers that each class include students with a range of academic abilities. One benefit of this policy relates to gifted students being taken out of class to be given enrichment in subjects that they excel in. Spreading such students out over all classes limits the impact on the class when they are taken out for enrichment activities.

When placing students manually, it is very difficult to simultaneously take all of these factors into account. An algorithmic approach to this problem allows administrators to decide how to weight these factors and determine class placements taking all of these factors into account. Our goal in undertaking the work described in this paper was to provided elementary educators with an easy-to-use Excel-based tool that helps them improve classroom assignments while spending less time on this task. Moreover, we wanted to provide a solution framework that could easily evolve and be fine tuned as needs dictate.

We present a model that was developed for helping the school to assign their students to classes based on the criteria provided above. The integer program described in this paper is used first to satisfy the basic constraints that the administrators determine are necessary for their class placements. It provides a starting point that can then be improved further using heuristics, especially if the integer program is found to have many feasible solutions. Next, because the school wishes to balance multiple objectives in creating classes, an evolutionary algorithm is used as a heuristic to further improve the class placements after the basic constraints are fulfilled with the integer program. Although the evolutionary algorithm is unsuitable for finding globally optimal class placements based on the inputs provided, it is generally suitable for finding a near-optimal solution that is satisfactory to the administration. It also allows administrators to easily tweak the class placements if they are unhappy with some characteristics of the model's outputs.

Before continuing, we briefly review some related literature. Cutshall et al. (2007) used integer programming to form student teams in the integrated core MBA program at Indiana University. Lopes et al. (2008) used integer programming to form student teams for senior design projects in the College of Engineering at the University of Arizona. Krass and Ovchinnikov (2006, 2010) used a heuristic column-generation approach to create MBA study groups in the Rotman School of Management at the University of Toronto. These problems all have the character of partitioning students into groups, balancing multiple objectives and a combination of soft and hard constraints. Of course the devil is in the details concerning the specific objectives and constraints, which vary quite a lot across these problems. One distinguishing aspect of our problem is that we are partitioning the set of students typically into a smaller set of larger groups. This kind of difference in the character of instances can have a dramatic effect on the solvability of partitioning problems, independent of whether we consider different objectives and constraints. This ultimately has some effect on the types of methods that were found effective in each of the applications. Finally, we mention that Aboudi (1986), Aboudi and Nemhauser (1991, 1990) developed integer-programming formulations and methods for a rather different classroom assignment problem based on an assignment-problem with side-constraints model. Although they were assigning classes to room/time slots, rather than students to classes, still the problem has the general flavor of an assignment type problem with side constraints. A distinguishing feature of their problem is that they were assigning *one* class to each room/time slot, whereas our problem as well as the other cited ones assign *many* students to each class.

We assume some familiarity with both linear integer programming and genetic algorithms, mostly at the modeling level. In §2, we give a linear integer-programming formulation that is aimed at generating a good feasible solution. In §3, we describe a genetic algorithm that we used to improve the solution generated by the linear integer program. In §4, we describe how we implemented our solution, and we summarize our computational results. We make some concluding remarks in §5.

2. Linear Integer-Programming Formulation

In this section, we give a linear integer-programming formulation of the problem. For an introduction to integer programming, see Wolsey (1998). We can regard the formulation that we give as a transportation problem with side constraints. Such problems, and more generally integer network-flow problems with side constraints have numerous applications (e.g., see the references in Barr et al. 1986, §1, p. 68).

From the viewpoint of formulating our problem as a transportation problem with side constraints, we regard each student as a unit of demand that must be satisfied exactly, and we regard each class as a supply point with a total supply equal to the capacity of the class. Next, we make this formulation precise, and as we go, we introduce all of the side constraints. We let m denote the number of students and n denote the number of classes. Let x_{ij} be a 0/1 variable with $x_{ij}=1$ indicating that student i is assigned to class j, for $i=1,\ldots,m$, $j=1,\ldots,n$.

For assigning each student to a class, we have

$$\sum_{j=1}^{n} x_{ij}=1, \quad \text{for } i=1,\ldots,m. \tag{1}$$

To insure that a maximum class-size κ is respected, we have

$$\sum_{i=1}^{m} x_{ij}\le\kappa, \quad \text{for } j=1,\ldots,n. \tag{2}$$

To make sure that every student is assigned to a class with one of their designated friends, we have

$$x_{ij}\le\sum_{s\in L_i} x_{sj}, \quad \text{for } i=1,\ldots,m;\ j=1,\ldots,n, \tag{3}$$

where L_i denotes the set of students comprising the list of friends of student i. Note how if student i is assigned to class j (whereupon the left-hand side of the inequality equals 1), then some student on student i's list is also assigned to class j.

Suppose that G is a group of students for which we want to make sure that none of them is the only one from their group in a class. This may not correspond to a set of friends selected by a student, but it could be, for example a geographic group. Then we can treat any such restriction similarly:

$$x_{ij}\le\sum_{s\in G\setminus\{i\}} x_{sj}, \quad \text{for } i\in G;\ j=1,\ldots,n. \tag{4}$$

To enforce that the boy/girl ratio is between p and q, we have

$$p\times\sum_{i\in F} x_{ij}\le\sum_{i\in M} x_{ij}\le q\times\sum_{i\in F} x_{ij}, \quad \text{for } j=1,\ldots,n, \tag{5}$$

where F is the set of girls and M is the set of boys.

If S is a set of students that must all be assigned to different classes, presumably with $|S|\le n$, then we can add the constraints

$$\sum_{i\in S} x_{ij}\le 1, \quad \text{for } j=1,\ldots,n. \tag{6}$$

Such constraints are normally called *clique constraints*, though that terminology here is ironic! We may have many of these constraints, but for computational purposes it is best to have these sets be large (as this implies a tighter linear-programming relaxation that can generally be exploited for efficiency by integer-programming solvers). For example, whether we use the single set $\{1,2,3\}$ or the three sets $\{1,2\}$, $\{1,3\}$,$\{2,3\}$, we will assign the three students 1, 2, 3 to different classes. But it is far better, computationally, to use the single set $\{1,2,3\}$ for defining the constraints.

In some situations, certain students *must* be placed together. These students form a clique in the more conventional sense. If S is a (maximal) set of students that must be placed together, we simply include the constraints

$$x_{ij}=x_{lj}, \quad \text{for all pairs } i\ne l\in S;\ j=1,\ldots,n. \tag{7}$$

Sometimes a student i must be placed in a specific class j. In such a situation, we simply fix $x_{ij}=1$ (and then we could remove x_{ik} from the model, for all $k\ne j$).

Sometimes there is a group of students $\mathcal{T}$ who are a bit too "energetic," finding it difficult to sit still in a classroom setting. We can easily limit the number of such students with constraints

$$\sum_{i\in\mathcal{T}} x_{ij}\le t, \quad \text{for } j=1,\ldots,n, \tag{8}$$

where t is the limit selected.

3. Heuristic

It is not so clear how to formulate an objective function. For example, matching up each student with one friend is considered important enough to be enforced via constraints, but each friendly match beyond that has some diminishing marginal value. Additionally, there really are multiple objectives that should be balanced (e.g., involving friend requests, academic diversity, etc.). Therefore, optimizing some linear objective on the assignment variables, subject to all of our constraints, may fall short of providing an ideal solution. As it stands, there may well be many feasible solutions to the constraints, and therefore we may expect a good deal of latitude in how we make the final assignments.

We used a genetic algorithm to improve on the feasible solution obtained via integer linear programming. Genetic algorithms are heuristics used for finding good solutions using processes that mimic the

process of natural evolution (see Eiben and Smith 2008 for an introduction). Such algorithms are well suited for heuristically handling nonlinear (nonconvex) objective functions that are not easily and efficiently handled in the context of linear integer programming. Of course there are other possibilities like employing off-the-shelf global-optimization software, but such methods have a very high computational cost. The genetic algorithm that we employed has a few distinguishing features.

- The genetic algorithm is seeded with one individual solution that was created using the feasible solution of the integer program, thereby speeding up the time to evolve an acceptable class placement.
- Rather than creating new generations entirely composed of individual solutions that went through the processes of crossover and mutation, the best individual solutions from each generation were copied into the new generations. This is known as the "elitism strategy."
- Each chromosome contains one gene for each student in the grade, and each gene is encoded with integer values between 1 and 4 to represent class placement in one of four classes.
- The fitness function, which is described further below, calculates how each candidate solution fulfills certain criteria (e.g., balanced boy-to-girl ratios, restriction on the number of overly energetic students per class, etc.).

Our fitness function is calculated using the following factors:

- Points are given for each of the parents' "friend requests" satisfied. To ensure that constraint (3) remains satisfied, the marginal value of satisfying additional requests decreases, with a particularly large drop-off after the first request is fulfilled. Specifically, zero points are given if no friend request is satisfied for a student, 7,000 points if one request is satisfied, 7,200 for two, 7,300 for three, 7,350 for four, and 7,375 for five. The specific choice of these numbers was determined in a back-and-forth process with administrators, learning what kind of solutions were desired.
- Points are also given for satisfying teachers' requests that certain students be placed together. In this case, the number of points given increases linearly with each request fulfilled. Specifically, we used 50 points per satisfied request.
- Points are subtracted for any violations of constraints that were previously satisfied with the integer program. In most cases, these penalties are linear and are large, making it unlikely that individual solutions that violate these constraints will reproduce. However, it is often the case in practice that these constraints are violated on occasion if doing so will improve the fitness function enough that the penalty is outweighed by the improvements. In the case of class size, this is encouraged by setting up the fitness function such that the penalty for deviating from the optimal class sizes increases exponentially as the class sizes diverge from the optimum. For example, if 100 students must be split into four classes of 25 (the "optimal placement" for this objective), a placement scheme where one of the classes has 26 students and another has 24 might evolve if it improves the class placements enough in other ways; however, greater deviations are increasingly less likely to evolve since the penalty for deviating from 25 students increases quadratically. Some specifics are as follows: A penalty of 100 $(25 - (\text{actual class size}))^2$; a penalty of 1,000 points for (i) each boy from Manhattan who is in a class without any other boys from Manhattan, (ii) each girl from Manhattan who is in a class without any other girls from Manhattan, (iii) each class that has fewer than six students from New Rochelle, (iv) each class that has only one student from the Bronx or Yonkers; a penalty of 1,000 $(0.6 - (\text{fraction of class made up of girls}))^2$ for each class where girls comprise less than 40% of that class; a penalty of 1,000 for each student who is designated as "too energetic" in excess of five in a class; for other constraints, we attach extremely high penalties to violations, so that these constraints are never violated.

4. Implementation and Computational Results

We implemented our algorithm in Microsoft Excel, using the OpenSolver[1] and Evolver[2] packages. Using Excel led to very rapid development of a useful implementation with a user interface that is easy for nonspecialists to use; see Jensen and Bard (2003) for further examples of this type of approach.

The data set used for testing the algorithm had the following characteristics:

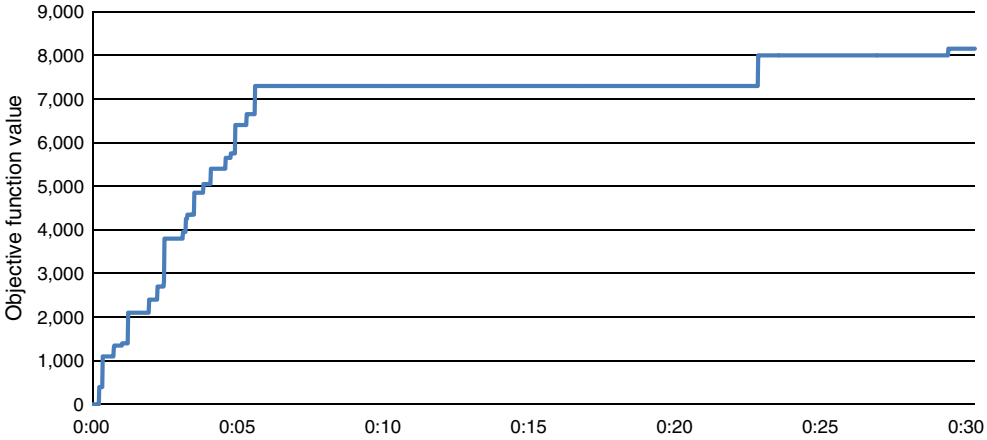
- 100 students who needed to be placed into classes of 25 students each.
- 43 girls and 57 boys (it was decided that a class may only have up to 66% boys, and preferably lower).
- 9 students made no requests for friends, 5 made only one request, 10 made two, 16 made three, 52 made 4 and 8 made 5 requests.[3]
- 39 pairs of students could not be placed together.

[1] opensolver.org.

[2] www.palisade.com/evolver.

[3] Every student's parents are asked to request four students. In some cases, the data set shows fewer requests, which is usually due to administrators eliminating some parents' requests before we got the data. Some parents will request more than four students, which is accepted by the school, since that makes it easier to guarantee at least one request. Some parents will make no requests, which gives even more leeway in forming classes.

Figure 1 Objective Value vs. Time (Minutes)



- 20 energetic students were in the grade, and it was preferable to divide them between classes as much as possible.
- 77 pairs of students requested (by teachers and other staff) to be placed together.
- 20 students needed to be placed into specific classes.
- 2 pairs of students and 1 trio of students had to be placed together.

After setting up the model, the OpenSolver package took approximately 2.55 seconds to provide a feasible solution for the first step of the process. The Evolver package was then used to further improve the class placements and was run for 30 minutes, during which time it made a small number of improvements. It was run with a crossover rate of 0.5 and a mutation rate of 0.1. A crossover rate of 0.5 implies that on average, each parent solution gives 50% of its genes to each offspring solution that it generates. A mutation rate of 0.1 implies that each gene has a 10% chance of being mutated. Because the genetic algorithm employed utilizes a stochastic process to create incremental improvements to the class placements, running the algorithm for longer generally gives better results, although the rate of improvement is slow. For this particular problem, 30 minutes was chosen as a reasonable amount of time to run the algorithm since improvements to the solution at this point were coming very slowly and the school administration was already satisfied with the available solution. Figure 1 shows the behavior of the objective value over a run of the algorithm. Note that a constant was added to the objective so that the initial value was zero.

5. Conclusions

SAR Academy successfully utilized our methodology to help assign students to classes in one grade for the 2012–13 school year, and they intend to utilize it further next year for other grades as well. So we can regard our modeling and our solution approach and delivery as very successful.

Although our model is useful for finding a good assignment of students, its results of course depend on having proper inputs as well as an effective set of weights for the genetic algorithm to use in balancing competing objectives. Therefore, it is important for the school to develop appropriate input methods and weights in order to use the model effectively in the future. After all, it is ultimately up to administrators to decide what constitutes a good class placement, so they must be involved in fine tuning the algorithm.

When inputting data, the administrators and teachers must be aware that the model needs correct and unambiguous data to effectively place students into classes. For example, when assigning students by hand, administrators may consciously or subconsciously assign varying degrees of importance to splitting up certain pairs of students. The current model could be modified so that it gives varying penalties for placing different students together (who should not be together), which might allow the model to be more flexible in improving placements in other ways. However, the model currently utilizes only constraints (6) to separate certain pairs of students, and these constraints treat all pairs in an identical fashion. A better model might use a tiered system, whereby only the more important pairs to be separated would be included in the integer program, and the genetic algorithm might address less important pairs. A system for properly inputting data in such a way that the algorithm could use it effectively is essential for using the model to create better placements.

As discussed earlier, there are multiple objective functions to be balanced in creating classes and a proper set of weights must be used to come up with the best solutions. Ideally, the most important issues could be dealt with using the integer program and

less important issues could be dealt with using the genetic algorithm (with large penalties for breaking constraints satisfied by the integer program). Furthermore, as much as possible, it is important for the administrators to assign weights to the various objectives so that the genetic algorithm can work toward better solutions. Of course, this is somewhat more of an art than a science, since it is ultimately up to the teachers and administrators to decide what constitutes the best placement. However, administrators can use the process of trial and error to assign numeric values to weight the importance of various factors, and since the model works relatively quickly, they can try out several solutions until they are satisfied.

Finally, other ways to accommodate the complicated nature of trading off multiple nonlinear objective functions include mixed-integer nonlinear programming or a more complicated linear integer program on an expanded set of variables. Though this would lead to a more difficult mathematical optimization model, it might provide better solutions. We leave investigating such directions as future research

Acknowledgments

The authors gratefully acknowledge that this paper would not exist were it not for the kind introduction (of Jon Lee and Daniel Newman) made by Stephen M. Pollock. The research of J. Lee was partially supported by the National Science Foundation [Grant CMMI—1160915].

References

Aboudi R (1986) A constrained matching problem: A polyhedral approach. Ph.D. thesis, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, New York.

Aboudi R, Nemhauser GL (1990) An assignment problem with side constraints: Strong cutting planes and separation. Gabszewicz JJ, Richard J-F, Wolsey LA, eds. *Economic Decision-Making: Games, Econometrics and Optimisation* (North-Holland, Amsterdam), 457–472.

Aboudi R, Nemhauser GL (1991) Some facets for an assignment problem with side constraints. *Oper. Res.* 39(2):244–250.

Barr RS, Farhangian K, Kennington JL (1986) Networks with side constraints: An LU factorization update. *Ann. Soc. Logist. Engineers* 1(1):68–85.

Cutshall R, Gavirneni S, Schultz K (2007) Indiana University's Kelley School of Business uses integer programming to form equitable, cohesive student teams. *Interfaces* 37(3):265–276.

Eiben AE, Smith JE (2008) *Introduction to Evolutionary Computing* (Springer, Berlin).

Jensen PA, Bard JF (2003) *Operations Research Models and Methods* (John Wiley & Sons, Hoboken, NJ).

Krass D, Ovchinnikov A (2006) The University of Toronto's Rotman School of Management uses management science to create MBA study groups. *Interfaces* 36(2):126–137.

Krass D, Ovchinnikov A (2010) Constrained group balancing: Why does it work. *Eur. J. Oper. Res.* 206(1):144–154.

Lopes L, Aronson M, Carstensen G, Smith C (2008) Optimization support for senior design project assignments. *Interfaces* 38(6):448–464.

Wolsey LA (1998) *Integer Programming*, 1st ed. (Wiley-Interscience, Hoboken, NJ).