Using Simulated Annealing to Solve the Discussion Groups Problem

David Castro
John Renze
Nicholas Weininger
Macalester College
St. Paul, MN 55105

Advisor: Karla V. Ballman

Introduction

Our task is to assign 29 corporate board members to a sequence of discussion groups organized into seven sessions, of which three are morning sessions led by senior officers and four are afternoon sessions not led by senior officers. We wish to find a combination of group assignments that best satisfies the following objectives:

- In the morning sessions, no board member should be in the same senior officer's discussion group twice.
- No group should have too many in-house members (there are nine in-house corporate employees among the 29 members).
- The number of times any two board members are in the same group should vary as little as possible.
- No two groups should have a large number of common members.

The specific case given assumes that all members will participate in all sessions and that there will be six discussion groups for each morning session and four for each afternoon session. Our model ought to be capable of producing good answers quickly under these assumptions and of adjusting to more general configurations of board members. In particular, it should be able to adjust to small changes, such as individual additions or subtractions of board members, without recalculating all assignments from scratch.

The UMAP Journal 18 (3) (1997) 307–320. ©Copyright 1997 by COMAP, Inc. All rights reserved. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice. Abstracting with credit is permitted, but copyrights for components of this work owned by others than COMAP must be honored. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior permission from COMAP.

Model Design Considerations

This problem is essentially one of optimization. The different potential assignments of board members form a solution space, and we must optimize the “fitness” of our assignments (how well they satisfy our objectives) over that solution space. Thus we have to consider the issues that come up when designing any optimizer: time and space efficiency, flexibility, and optimality of the solution. We also have concerns particular to this problem.

Multiple Objective Satisfaction

Most optimization problems involve maximizing a single objective function; here we have four objectives. The problem statement doesn’t tell us whether, for example, to consider minimizing common group membership more important than minimizing the number of times any two board members meet. Presumably, we want to use some sort of weighted combination of the objectives as the function to optimize over the solution space—but determining how to combine and weight them is nontrivial.

Large Solution Space

The number of ways of assigning the board members to the discussion groups is enormous. Since each board member goes into seven sessions, we have a total of $29 \times 7 = 203$ variables; since there are six possible assignments for each member in the morning sessions and four for the afternoon sessions, the total number of possible solutions is on the order of $6^{87} \times 4^{116} \approx 3 \times 10^{137}$. Furthermore, no matter how we define our objective function, the solution space will probably contain a large number of local optima. This has two implications. First, it’s probably impossible to find a global optimum (and it isn’t absolutely necessary here). Second, we need a solution model that doesn’t require searching over a significant fraction of the solution space.

Fast Readjustment

Since board members may drop out or in at the last minute, we want a way of taking a precomputed solution and adjusting it for small changes in the member configuration. This adjustment should be significantly faster than a complete recomputation and should also involve fewer changes of assignment.

Simulated Annealing

We chose simulated annealing as the basis for our model, as it offers the best chance of constructing an effective solution-finding algorithm.

The Simulated Annealing Process

The simulated annealing algorithm can be described as follows:

1. Start with an objective function that is evaluable for every point in the solution space, a randomly chosen point in that space, and an initial "temperature" value.
2. Evaluate the initial point's objective function value.
3. Perturb the point by a small amount in a random direction.
4. Calculate the objective function value of the perturbed point.
5. If the new function value is better than the old one, accept it automatically, staying at the perturbed point.
6. If the new function value is worse, decide probabilistically whether to stay at the perturbed point or go back to the old one. The probability of not staying at the new point should depend on how low the temperature is and how much worse the new point is than the old one.
7. Lower the temperature slightly and go to step 3. Iterate until the temperature is close to 0 or the solution stops changing.

The algorithm essentially does a hillclimb through the solution space, with occasional random steps in the wrong direction. The temperature controls how likely the algorithm is to take a step the wrong way; at the beginning, the algorithm jumps almost completely randomly around the solution space, but by the end it almost never takes a wrong step. The idea is that the random steps allow the hillclimber to avoid getting stuck at local minima or maxima. In most implementations of simulated annealing, the probability of acceptance of a wrong-way step is $e^{-d/T}$, where d is the difference between the old and new objective functions and T is the temperature. The temperature typically starts at a value sufficient to give almost any degree of wrong-way movement a significant probability of acceptance, and decreases exponentially from there.

The original idea for simulated annealing comes from statistical mechanics. The molecules of a liquid move randomly and freely at high temperatures; when the liquid is cooled and then frozen, the molecules essentially "search" for the lowest energy state possible. The minimum energy state occurs when the molecules are arranged in a specific crystalline structure. If you cool the liquid slowly, the molecules will have time to redistribute themselves and find this structure; if you cool it quickly, they will typically get "stuck" at a higher-energy, noncrystalline state.

Reasons to Use Simulated Annealing

A number of factors influenced our decision to apply simulated annealing to our problem:

- It's fast. Simulated annealing requires evaluation of the objective function over a relatively small number of points to achieve its result.
- It's simple. Everything except the fitness function evaluation can be done in well under 100 lines of C code. It isn't difficult to understand or debug.
- It lends itself well to discrete solution spaces and nondifferentiable objective functions. As long as you can provide a random jump between solution points and a way of evaluating the function at any point, it can work; many other methods require a continuous solution space, or demand that you evaluate the function's derivative.
- It has a track record of success. Simulated annealing has been used for applications as diverse as stellar spectrum analysis and chromosome research.
- We were all well acquainted with simulated annealing, and one of us had previously used it to solve a similar partitioning problem.

Alternatives

Heuristic Algorithms

One could try to get a solution by using some sort of intuitive rules about how to rearrange things, much as a human secretary would. This would likely be too slow for a large number of members, though, and coming up with good intuitive rules that a computer can implement is difficult.

Gradient Methods

Traditional gradient descent methods are another possibility. These, however, generally require that the objective function's derivative be evaluated or at least estimated, and our objective function is extremely difficult to express mathematically. Furthermore, they run the risk of getting stuck at local minima.

Integer Programming

Integer programming is commonly used for discrete optimization problems like this one. But none of us had experience implementing it, and we feared that the large size of the solution space might make it too slow.

Genetic Algorithms

We could establish a pool of potential solutions that would "evolve" toward an optimal solution through a process analogous to natural selection. But this, too, would likely be too slow and complex for our problem.

Modeling the Solution

We designed and coded in C a program that uses simulated annealing to solve a general set of discussion-group problems, including the one given.

The Data Structure

Our approach to the problem began with data structure design: We needed a way to encode the whole list of assignments that would allow us to code the annealing process straightforwardly, and preserve important testing and organization data for analysis of the annealing results.

The data structure `partition` has three levels of organization:

1. The top level, `partition`. This contains a lot of data pertaining to the whole solution structure: the number of morning and afternoon sessions, the number of groups in each type of session, and the total number of members and of in-house members participating in each session. It also contains some data relevant to the objectives concerning pairs of members and common membership (more on that below).
2. The list of groups contained in the `partition`. Each `group` corresponds to one of the discussion groups in one of the sessions and has a size variable.
3. The list of people contained in each `group`. Each `person` has a name, a number and a flag indicating whether they're in-house. Thus each instance of `person` corresponds to one member's participation in one session. Note that this allows members to participate in only some of the sessions.

The Objective Function

We chose as the objective function a weighted sum of four subfunctions. Each subfunction takes a partition and calculates how close it comes to satisfying one of the objectives given in the problem. The results are all expressed in terms of "badness," or the degree to which the partition fails to satisfy an objective; thus our problem becomes the minimization of the "badness function."

Senior officer nonrepetition

The first objective we consider is that no board member should be in the same senior officer's morning group twice. Our first subfunction simply takes the morning groups headed by each senior officer and checks for repetitions in their member lists. The total number of repetitions—that is, the total number of times a board member is in the same senior officer's morning group more than once—is multiplied by one weight to form the first part of the badness function. Since we want zero repetitions, and want the annealer to stay with solutions with zero repetitions once it finds them, we also have a “zero bonus” that subtracts a given amount from the badness if there are zero repetitions.

In-house members

The second objective states that no group should have a disproportionate number of in-house members. Our second subfunction calculates the amount of disproportionality in the distribution of in-house members among groups. For each session, it uses the number of total members and in-house members participating in the session to compute ideal floor and ceiling values for the number of in-house members in each group. (If the number of in-house members is a multiple of the number of groups in the session, then the floor and ceiling are the same; otherwise they differ by one.) Then it goes through each group in the session and tests to see if the number of in-house members is more than the floor or less than the ceiling; if so, it adds to the disproportionality sum the difference between the floor or ceiling and the actual number. So, if the ideal ceiling were two in-house members per group, a group with four in-house members in it would add two to the disproportionality sum.

We want the disproportionality sum, like the repetition sum, to be zero, and so implement a zero bonus; we set this equal to the disproportionality weight rather than making it separately adjustable, for reasons discussed below.

Pairings of board members

Our third objective is to try to ensure that each board member meets each other board member approximately the same number of times. Our third subfunction does this by computing ideal floor and ceiling values for that meeting number, just as the second does for the proportion of in-house members. In the `partition` structure we keep a matrix whose elements correspond to the number of times each pair of board members are in the same group. The third subfunction recalculates this matrix by looking at each pairing in each discussion group; derives the ideal floor and ceiling values from the matrix; and then goes through the matrix again, adding up the instances in which the number of times a pair of members meets is less than the floor or more than the ceiling. This sum becomes the *pairwise anomaly count*.

We also calculate the maximum number of times any two board members meet, and use that as well as the pairwise anomaly count in the badness calculation. The reasoning behind this is not only because it’s desirable for most pairs of members to meet the same number of times, but also to ensure that no pair of members is in a hugely disproportionate number of groups together. We don’t want a solution with one pair that is in every group together but with no other anomalous pairs to be preferable to a solution with many slightly anomalous pairs but with no hugely anomalous ones.

Here, we have no zero bonus for the anomaly sum, because the meeting criterion ought to be satisfied “as much as possible,” rather than absolutely, and because it’s impossible in many cases (including ours) to drive it to zero.

Common membership

Our final objective is to ensure that no two groups have a disproportionate number of common members. The fourth subfunction is almost precisely analogous to the third. Again, we have a matrix in the `partition` structure, listing for each pair of groups how many members they have in common; again, we calculate this matrix, use it to derive an ideal mean, and find the sum of deviations from that mean and the maximal deviation. Here, however, we don't count as deviant groups those that have fewer than the mean number of common members. It doesn't matter if two groups have no members in common, only if they have too many.

The Annealing Iteration Process

Once we have the "badness function" described above, we perform simulated annealing. The relevant implementation details are:

- The perturbation: A single swap of two members is the unit of random perturbation. Our program randomly chooses a session, two discussion groups within that session, and a member in each group, and then swaps them.

- The initial configuration: The file-reader dumps the members from the file into the `partition` in order, giving an extremely bad configuration. Our program does random swaps on that configuration for a random number of times (between 1 and 32,767), producing a "shuffled" initial configuration.

- The starting temperature setting and the rate of exponential decay: These are two key variables that determine how long the annealing takes and what the temperature profile is.

  For the starting temperature, we just use the badness value of the starting configuration; this means that at the starting temperature, a solution as bad as our starting one would have a 10% chance of being accepted as a step away from a perfect (zero-badness) configuration.

  For the decay, we used a variety of different rates. Multiplying the temperature by 0.998 at the end of each iteration tends to give the best time/accuracy tradeoff. Slower decay makes for long annealing runs that don't get much better; faster decay doesn't give the process enough room to "jump around" randomly at the beginning, resulting in much worse solutions.

Setting the Weights

We anneal in an attempt to minimize the following badness function:

$$\begin{aligned}\texttt{badness(partition)} &= w_0 * \texttt{reptotal} - w_1 * \texttt{repzero} + w_2 * \texttt{disprop} \\ &\quad - w_2 * \texttt{diszero} + w_3 * \texttt{pairanom} + w_4 * \texttt{maxpair} \\ &\qquad + w_5 * \texttt{commanom} + w_6 * \texttt{maxcomm}\end{aligned}$$

where

- `reptotal` is the number of times a member is in the same senior officer's group twice;
- `disprop` is the disproportionality sum for in-house members;
- both `repzero` and `diszero` are 1 if `reptotal` and `disprop` both are 0, and both are 0 otherwise;
- `pairanom` and `maxpair` are the anomaly score for pairwise meetings and the maximum number of times a pair meets;
- `commanom` and `maxcomm` are the anomaly score for group common membership and the maximum number of members a group has in common; and
- $w_0, \ldots, w_6$ are integer weights.

How should we set the weights? We did so by trial and error. The set of weights $w_0 = w_1 = 1200$, $w_2 = 1000$, $w_3 = 400$, $w_4 = 4000$, $w_5 = 100$, and $w_6 = 500$ produces extremely good results for a 9,000-iteration annealing run on the standard problem configuration. Such a run takes about 5 min on an HP 712/60 workstation. Using these weights, we could repeatedly produce solutions that had no senior officer repetitions, no instances of in-house member disproportion, no pair of members that met more than three times, and no pair of groups with more than three members in common.

We used somewhat different weights for a longer annealing run to produce our very best solution, and also adjusted the weights to produce solutions for different session configurations, as we will describe below. But the set of defaults above appeared to work remarkably well for a variety of configurations.

Some things to note about the default weights:

- The zero bonus for senior officer nonrepetition equals the minimization weight. That works so well that we hard-coded in the same equality for in-house disproportionality, rather than making another adjustable zero bonus.
- The ratio of pairwise minimization weight to pairwise maximum weight is 1 to 10. That means that the algorithm considers reducing the pairwise anomaly score by 10 equivalent to reducing the pairwise maximum by 1. Lower ratios tend not to force the pairwise maximum low enough; higher ratios tend to prevent the annealer from occasionally making the pairwise maximum higher by 1 on its way to a much lower pairwise anomaly score.

- The common membership weights are quite small. We found that good common membership configurations were strongly correlated with good pairwise meeting configurations; that is, configurations in which no pair of members met an inordinate number of times also tended to be configurations in which no two groups had too many common members.

Our Solution

Using extra-long runs and adjusting the weights by trial and error, we eventually produced the solution in **Tables 1–2**. The in-house members are 1–9 and the non-in-house members are 10–29.

Table 1.
Assignments by discussion group.

Session	Group	Members
Morning 1	1	27 22 20 17 1
	2	4 28 21 3 13
	3	10 11 8 29 18
	4	14 19 24 7 23
	5	15 9 5 16 12
	6	6 25 2 26
Morning 2	1	13 29 2 6 28
	2	10 16 9 19 25
	3	7 23 20 22 12
	4	5 11 26 17 3
	5	8 14 20 27 4
	6	24 21 1 15
Morning 3	1	16 8 19 21 5
	2	2 18 22 1 23
	3	15 26 17 9 14
	4	12 25 4 29 27
	5	2 13 20 11 6
	6	28 7 3 10
Afternoon 1	1	27 26 24 2 8 10
	2	6 3 22 15 19 4 18
	3	11 16 23 5 25 14 1 28
	4	21 9 13 17 29 7 12
Afternoon 2	1	23 6 5 21 10 27 12
	2	3 15 29 14 2 25 20
	3	26 9 1 8 28 19 22 13
	4	7 24 4 17 16 11 18
Afternoon 3	1	15 4 1 13 2 10 16
	2	5 24 12 25 22 17 8
	3	21 28 14 20 7 26 18 6
	4	27 23 11 19 3 9 29
Afternoon 4	1	7 25 27 13 19 5 18
	2	21 14 22 11 10 2 9
	3	12 29 3 6 16 24 26 1
	4	4 23 17 15 28 20 8

Table 2.

Assignments by board member.

	Morning			Afternoon			
	1	2	3	1	2	3	4
In-house members							
1	1	6	2	3	3	1	3
2	6	1	2	1	2	1	2
3	2	4	6	2	2	4	3
4	2	5	4	2	4	1	4
5	5	4	1	3	1	2	1
6	6	1	5	2	1	3	3
7	4	3	6	4	4	3	1
8	3	5	1	1	3	2	4
9	5	2	3	4	3	4	2
Other members							
10	3	2	6	1	1	1	2
11	3	4	5	3	4	4	2
12	5	3	4	4	1	2	3
13	2	1	5	4	3	1	1
14	4	5	3	3	2	3	2
15	5	6	3	2	2	1	4
16	5	2	1	3	4	1	3
17	1	4	3	4	4	2	4
18	3	5	2	2	4	3	1
19	4	2	1	2	3	4	1
20	1	3	5	1	2	3	4
21	2	6	1	4	1	3	2
22	1	3	2	2	3	2	2
23	4	3	2	3	1	4	4
24	4	6	5	1	4	2	3
25	6	2	4	3	2	2	1
26	6	4	3	1	3	3	3
27	1	5	4	1	1	4	1
28	2	1	6	3	3	3	4
29	3	1	4	4	2	4	3

How Good Is This?

In this configuration, no member is ever in the same senior officer's morning discussion group twice. No group contains a disproportionate number of in-house members (the morning groups contain 1 or 2, the afternoon groups 2 or 3).

No pairs of members are in the same group together more than 3 times. Of the possible pairs of members, 40 never meet; 214 meet once; 138 meet twice; and 14 meet three times. Thus the pairwise anomaly score is 54, and the vast majority of members meet one another a "reasonable" number of times (the mean number of meetings is about 1.3). It is possible to achieve a configuration in which no two members meet more than twice, but not while preserving the other objectives.

Also, no two discussion groups have more than two members in common.

This is as low as possible.

We conclude that this configuration satisfies all the objectives well and three of the four perfectly; it is likely extremely close to the global optimum.

Typical Results on the Standard Configuration

A typical annealing run, with default weights, will zero out the senior officer repetition and in-house disproportionality. It will also reduce the maxima of member-pair meetings and group common membership to 3. It will not usually reduce maximum group common membership to 2, and it typically gives a pairwise anomaly score of 50 to 60.

Thus, the standard annealing run's result isn't quite as good as our best; but it's nearly as good and much easier to achieve. It considers only about 9,000 different sets of assignments in reaching its solution, and takes about 6 min.

Generalizing the Model

More and Different Board Members

We tested the annealing process on data sets with a variety of different numbers of board members, ranging from 20 to 100. We also tried keeping the number of members at 29 and adjusting the proportion of in-house members. Finally, we tried keeping the existing set of 9 in-house and 20 other members but changed the number of senior officers, the number of morning and afternoon sessions, and the number of groups in each afternoon session. In all cases, we annealed with the same default weights used for the standard configuration.

The model responded extremely well in each case. **Table 3** shows the times required for a full annealing run on the standard session configuration with various numbers of members.

Table 3.
Run-time results.

Members			Run-time
In-house	Others	Total	
5	10	15	2:56
9	20	29	6:38
12	27	39	8:25
15	33	48	9:53
18	41	59	22:06
30	70	100	58:01

We also tried changing the profile to 4 in-house and 25 other members, and to 14 in-house and 15 other members. In both cases, the annealing still drove the in-house disproportionality to zero and reduced the pairwise and commonality maxima to 3.

We tried a total of five different changes in the session / group profile; these ranged from decreasing the number of groups in each morning session to two to having only one morning session and six afternoon sessions. In almost all cases, the annealing produced solutions that were as good as could be expected, given the limitations of the sessions and groups.

The only exception occurred when we tried having three officer-led groups for each of the three sessions. Then the default weights failed to minimize the senior officer nonrepetition criterion, which is much harder to achieve with three groups than with six. Increasing w_0 to 2,000 and rerunning gave much better results.

Different Levels of Session Participation

The data structure and annealing code are designed to deal transparently with members who attend some but not all of the sessions, by considering each member's participation in each session as a separate variable. We tested our code on several different session-participation variations, including:

- making the in-house members not attend the afternoon sessions,
- making some of the non-in-house members not attend the morning sessions, and
- introducing new members who go only to one morning session.

In all cases, the annealing produced good results—zero senior officer repetitions, no more than two instances of in-house disproportionality, maximal pairwise meetings, and group common memberships at one more than the ideal mean.

Adjusting Quickly to Last-Minute Changes

Another important consideration is how well our model can deal with small changes in the configuration of board members—one or two new board members added or deleted, say. We investigated two approaches to adjusting an already-annealed configuration, reannealing and flip-path search. Both are motivated by the idea that the new configuration's best solution should be quite close to the old one's. We tested these approaches on several small modifications: single and double additions and deletions to all sessions, plus additions to only one or two sessions.

Reannealing

Since we want to stay in the neighborhood of the old configuration and to take less time than the original annealing, we use a much lower starting temperature. Experimentation shows that dividing the regular starting temperature by 1,000 works best. The new configuration often is very different from the old one. This would be undesirable in real-world applications, where you might want to make as few changes to group assignments as possible.

Greedy path search

The alternative is to try to reduce the badness function with as few changes as possible. One way is to try all possible single flips in the configuration, then try all possible combinations of two possible single flips, and so on, always looking for the flip sequence that produces the lowest badness function value.

This approach quickly becomes too slow. There are 2,310 possible flips in the standard configuration, and evaluating all of them takes more than 1 min on an HP 712/60; thus, evaluating all two-flip sequences would take at least 20 h. An alternative, much faster approach is to try all single flips, take the one that produces the lowest badness function, perform that flip, try all single flips from the resulting flipped configuration, and so on. We call this a *greedy path strategy*, and in our tests it brought the new configuration's badness function down acceptably close to the old one's in seven to ten flips.

Furthermore, we observed during our tests of the greedy path search that it tended to do best after doing one flip involving groups in each of the seven sessions. This was especially true when we added a new member to all of the sessions. This makes sense, because it ought to take one flip to put a new member in the "right" place in each session.

So we tried the following variation: Perform the best of the possible flips involving groups in the first session, then perform the best of the flips involving the second session, and so on, to the last session. This requires considering all of the 2,310 possible flips only once. This does not work nearly as well as the original greedy search, probably because this approach fixes the order in which the best flips in the sessions are taken.

We finally found a workable hybrid approach. This approach starts out by finding and taking the best flip from all the sessions, then takes the best flip in each session in order, and finally again finds and takes the overall best flip. It runs in roughly 5 min for the standard configuration and gives test results about as good as for the original greedy path search. It doesn't always work as well as reannealing, but it never requires more than nine (in the standard configuration) single-flip changes.

Improving the Model

Complexity

Our algorithm runs in time approximately proportional to the square of the number n of board members. The pairwise part of the badness calculator goes through a matrix including all $n(n-1)/2$ pairs of members. The run-time is also quadratic in the number of groups, because of the pairs-of-groups common membership matrix.

It would be nice to develop a badness function that runs in linear time and produces a good approximation to our original function value. We could also make smaller efficiency improvements, such as developing a way to update efficiently the pairwise and commonality matrices with each single flip instead of recalculating them on each iteration. Time considerations prevented us from doing this (we tried doing it with the pairwise matrix but never got it to run significantly faster than straight recalculation). We believe that our algorithm runs remarkably well as it is; finding a near-minimum over a solution space of 10^{137} points in 6 min is no small task.

Flexibility

The present model allows only two kinds of group sessions—morning and afternoon—and assumes that all sessions have the same number of groups of the same size (or differing by only one). It wouldn't be too difficult to extend the `partition` structure to allow for a more complex session structure, such as one that would allow for different numbers of senior officers at each morning session.

We could also add different types of board members and specify new objectives based on them. Perhaps, for example, one might want to stipulate that some board members are new, and that new board members should all be in the same discussion groups so that they can get to know each other (or at least that every new member should meet every other member once).

Finally, we could try to devise a general method for setting annealing weights, given a configuration. The default standard weights appear to work well for a large range of configurations, but they are almost certainly not optimal for all configurations.

References

Kirkpatrick, S., C.D. Gelatt, and M.P. Vecchi. 1983. Optimization by simulated annealing. *Science* 220 (13 May 1983): 671–680.

Press, William, et al. 1992. *Numerical Recipes in C.* 2nd ed. New York: Cambridge University Press.