**Web Application: Event Attendee Scheduling Based on Mutual Availability**

Pat Gehrke & Basis Publishing, LLC

We are looking for an app that will provide an optimal schedule of event attendees based on their mutual preferences for dates and times. The app user would be the event organizer. She might upload a spreasheet of 100 attendees who signed up for a workshop with 35 possible date & time options. The app user wants to hold up to 10 maximum workshops with each workshop hosting at least 5 and no more than 7 attendees.

The user uploads the spreadsheet with attendee availability into the application, changes the application settings to minimum of 5 attendees, maximum of 7 attendees at each event, with no two events scheduled for the same time (or selects an option to allow duplicates), and then clicks submit. The software returns a downloadable file with all 100 attendees sorted into groups of 5-7, reflecting the optimum possible distribution based on attendees' preferred times. If a schedule that fits the parameters cannot be created, the app returns the best possible arrangement and a list of names who could not be sorted into date and time slots.

We and our clients have been doing this kind of scheduling manually for three years and would like to automate the process. We use it for scheduling webinars. Our clients are university professors who use it for scheduling online meetings, study sessions, and class presentations. Imagine having 100 students and you want to run 1-hour study sessions in groups of 8-12 over four days. You might have 30 timeslots that could work for you over those four days, but you don’t want to dedicate 30 hours holding study sessions. So, you poll your students and ask each to rank their 6 most preferred day/time slots. Based on their responses you then have to find 9-10 timeslots that fit everyone’s schedule, keeping the groups in the 8-12 attendees range, and ideally finding a schedule so students get higher ranked choices (optimal vs. simply functional).

Details on the data sets we use:

This application should sort people into scheduling options based on mutual availability. App users will upload a spreadsheet containing the following columns: Name, [Date&Time 1], [Date&Time 2], [Date&Time 3]..., out to perhaps as many as 35 date & time options. In rows below the header row the name column contain an attendee’s name. For each name listed 6-8 date & time columns will be ranked 1 (most preferred) to 8 (least preferred) and then hold blanks for the remaining columns.

For example:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Mon 9/1 10am | Mon 9/1 Noon | Tue 9/2 10am | Tue 9/10 Noon |
| Peter Parker | 1 |  | 2 |  |
| Harry Osborn |  | 2 | 1 |  |

The spreadsheet would continue down for perhaps as many as 100 possible attendees (or as few as 20) and out to perhaps as many as 40 date & time options (or as few as 15). We will provide two real-case data sets in CSV or XLS/XLSX format for development and testing.

The app user will upload a CSV or Excel file and tell the app the minimum and maximum attendees that should be scheduled for each event and whether duplicate events within one timeslot are acceptable. The program then attempts an optimal distribution of attendees into the options that work best for them.

How we solve the problem now:

We'd normally have at least 20 attendees sign up, more commonly 45-50, but sometimes up to 100. The schedule options usually extend out to 30 or 40 options. In some cases attendees rank up to 10 options, but usually we ask they rank 6 or 8.

The rank numbers attendees enter reflect their rank-order preference (1st choice, 2nd choice, etc.).

What we want is to give as many people their highest rank choice as possible, while still ensuring that everyone gets scheduled. We're constrained though by needing sets of a given size (varies from event), so we can't just give everyone their first choice. So, we need to sort the names into groups of a set range (like 4-6) that reflect the best distribution to provide everyone a timeslot and give people their highest ranked slots possible.

Here's how we solve it by hand on a very large whiteboard:

1. Make a grid of all the day/timeslot options (30 day/timeslots would be a 5x6 grid). Each box is labelled to reflect one day/timeslot.
2. Starting with the first name, we write their name and preference number in each corresponding box. So, in the example above we write "Osborn-2" in the Mon 9/1 Noon box, "Black-1" in the Tue 9/2 10am box, and so on, until all the preferences for every person are on the whiteboard. Some timeslots have a ton of names written in their boxes, some boxes have almost nothing.
3. Step back and look at the whiteboard. We immediately will spot timeslots where too few people listed it as an option. These are not viable parts of the solution (cannot meet minimum number of attendees) so we eliminate them.
4. We then look for people whose first and possibly second choice (or even 3rd, etc) are eliminated by step 3. We are looking for cases that appear to be very difficult to find solutions where the person gets a timeslot in a group of the minimum size -- these are people who choose mostly unpopular timeslots. Usually one or two people are in this situation. Usually this leads us to a situation where someone gets a "lower" ranked choice as their only viable solution and that solution requires we lock in other people to make that timeslot meet its minimum number of attendees.
5. So now we know we likely have to use that timeslot and look at the other people in that timeslot box, giving it to the people there who ranked it the highest, but also looking at how that will affect the other boxes they are listed in (will giving this slot to Smith, mean another viable slot no longer has enough recipients, etc.). This sometimes causes a little reshuffling, but usually we have 1 slot and group of attendees locked in at this point.
6. Cross-out the names of attendees now locked into that slot from any other slots they had ranked.
7. Go back to step 3 and repeat until finished.

Platform requirements:

We need a web-based application that can run on our servers. We have a commercial VPN (KnownHost) running CentOS 7.6 with Apache 2.4.39, MYSQL 5.7.27, and PHP 7.0 (we will be upgrading to a newer version of PHP in December). The user interface should also be web-based.

We are happy for the code to be released under an open source license that allows us to use it for our clients and business partners.

Interface requirements:

1. Web-based.

2. Ability to upload a csv file with attendee preferences (see sample data set for format). Excel file compatibility would be nice, but isn’t essential.

3. Input screen that allows the user to set parameters for the sorting each time:

Minimum Group Size?

Maximum Group Size?

Allow multiple groups in the same day/time slot?

Maximum Number of Groups?

4. If the program cannot find a solution, ideally it will provide the closest solution it can produce and a list of the people it could not schedule. (While not being able to schedule everyone into one of their selected slots is possible, it has never happened to us. In fact, we rarely have to go below anyone’s 4th choice.)

Additional thoughts/ideas:

“Optimal” in this case is based in their ranking of preferred days/times.

For example:

Solution 1: Smith gets 3rd choice, Williams gets 2 choice, Davis gets 1st choice, Maxwell gets 6th choice

Solution 2: Smith gets 3rd choice, Williams gets 3rd choice, Davis gets 2nd choice, Maxwell gets 3rd choice

Solution 2 is preferable to solution 1.

To determine relative optimality, you could just use a simple sum of the rankings in a solution. The solution with the lowest score would be the most preferable option.

It might be nice though to weight the lower options a little more heavily, so that moving past the 4th option would be increasingly disadvantageous to the score of the solution. Something like this would be great (but isn’t essential):

|  |  |  |
| --- | --- | --- |
| Rank | = | Score |
| 1 | = | 1 |
| 2 | = | 2 |
| 3 | = | 3 |
| 4 | = | 5 |
| 5 | = | 7 |
| 6 | = | 10 |
| ≥7 | = | Rank\*2 |

Again, the solution with the *lowest* total score would be the most optimal solution.

The solution doesn’t have to be *truly* optimal, but near-optimal would be fine. The most important thing is that everyone is scheduled within the parameters (if this is possible) but near-optimal would be enough. We realize that true optimality is difficult to ensure (and I doubt we are accomplishing it now with our manual sorting.)