About Our Java Algorithm

Howard Lin, Randy Chen, Cody Abe

April 26, 2017

How our algorithm works

When given a .txt file of all individuals followed by their attributes (Which currently are schedules and languages), such as the following:

```
2
Howard Lin
AM,AT,AC,BM
JAVA,C
Randy Chen
AM,AT
JAVA,PYTHON,C
Cody Babe
CM,CT,FF
JAVA,PYTHON,C
Dat Man
FSU
C++
Jack Broccoli
AM,BW,BF,BSU
JAVA
Hugh Heffner
FSU,BM,BT,BR
C++,C
Dat Ho
```

the first line will determine the number of types of attribute (so in this example, we have 2: Schedule and Languages), and our Java implementation will record each individual, save their name and attribute details as a *person* object, and then our implementation will sort the list of individual by availability in schedule and people with less openings in their schedule will get priority. Then, in a greedy fashion, it will go through the list in order, and we check for number of commonalities in their attributes, and if they have more than a certain number of commonalities they are considered to be "group compatible", and then thrown in a group. Our implementation will then output a text file of the groups, to which our site will read and output. Any other available people not grouped (say there are two people left for groups of three) will just be added to random groups to form group of size $n + 1$, where n is the group size, which our implementation has set to 3.

Currently Known Drawbacks

Currently, here are the known issues that our implementation has:

- Our implementation is very greedy, and favors people of few availability so much to the point that they may be grouped with people who can only pair with certain people, despite the people of few availability being able to pair with other people. *e.g.* Person a who has little availability may be paired with person b, but could have also been able to pair with person c. However, person d is only compatible with person b, which is no longer available.
- The last few people that the implementation will try to find pairing with may just be randomly grouped if they cannot find another group. This may be bad since they may not work out. We tried to implement a reordering, but this was met with bugs and we worry that we could not get something work in on time. This may be something we will focus on in the future iterations.
- No adjustable "weights" for matching (see "**Planned Future Features**")

Planned Future Features

Currently, these are what we planned to add, and have setup (but not completed yet) for the project if we do plan to proceed with it:

- **Regenerating groups**. If the user is not satisfied with the group generated (manually by user) or if the groups made are not optimal (automatic by implement). The later would be tricky, and an idea that we had would be to try multiple combinations of groups and comparing each group sets to see which one would hold the highest "optimal" groups.

- **Adjustable weights for users**. Right now, our implementation is actually set up in a way that already takes adjustable weights into account (groups have a max amount of common languages and availability so that other groups have a chance of pairing with them, etc.). The trouble at the moment is linking it to the website.