Library used: copy, random, network, multiprocessing, pickle, time, itertools, matplot, os and priority queue from: https://github.com/mjwestcott/priorityqueue
Our algorithm is basically a greedy+random algorithm. The core is a value function to determine the local gain when we add a vertex into a bus. The value function is defined:

1. V= set of all vertices in G
2. B = set of all vertices in a bus that currently we try to fill
3. Y = total edges from v to set V\B
4. X = total edges from v to B
5. w = weight variable, determined from sampling and regression

value is X–Y*w.

1. We first determine an approximate number of nearly full buses, and the rest of the buses are "garbage buses", each of which is assigned a student with fewest friendships (Intuition: prioritize fully filling some buses to increase score.)

2. We initialize a max priority queue, and initialize the key to rank every vertices, v, to be a tuple (X–Y*w, X), where X–Y*w is calculated by value function, and X is for tie-breaking. Initially, there are no vertices in any buses, so we initialize the rank tuple to be (–degree(v), 0); that is, we start to process vertices with the smallest degrees in order to maximize the number of vertices with large degrees not in any buses when the buses is nearly full (Intuition: vertices with larger degrees have higher probability as good "patches" to fit in any buses).

3. We pop from the priority queue, put into a bus, and update its neighbors' rank. We try to fill a bus as full as possible before considering another bus, which is consistent with intuition in step1. If adding a vertex v to the bus will form a rowdy group within the bus, we ignore it, and process next vertex that pq pops.

4. When we switch to an anther bus, we need to re-initialize the pq as we did in step2, because our value function is bus-dependent.

We try to deal with vertex that will cause a rowdy group in a bus. We consider another variable, Z, the number of edges becoming invalid when we put this vertex into the bus, and replace the value with X–Y–Z. This approach runs very slowly so we gave up. Another attempt is to multiply a coefficient, w on Y, and we use linear regression to find the optimal value for w. Also, we refined some of our outputs by randomly swapping and moving students, and keep the modification if score improves, which fix the situations that our assumptions don't work well on specific inputs.

5. After using greedy, which can achieve ~46% overall score in 5 minutes, we ran out random swap algorithm on small and medium to improve the result, this gives a ~4% boost but takes much longer time. We left it running for many iterations which takes hours

We wrote multi-threading with python. For small, medium, large, it takes 103 seconds on Xeon E5 2673 v3 QS 12C/24T 2.4GHz - 3.1GHz
The time is count for solving only, data loading and saving time is not included.