# CS170 Project Report - Deliverable 2

Colin Lai & Austin Chang

We approached the design problem of making a good solver by thinking of situations where a single algorithm that generates an optimal solution in some graphs, could in others generate suboptimal solutions. In deliverable 1, we based our inputs on such edge cases, and when designing our solvers, decided to use multiple approaches and return the maximum scoring output between them.

The first algorithm is based on k-way greedy partitioning[1] which fills buses in a greedy fashion. We first filled buses with single rowdy groups and took those buses out of consideration for the main algorithm. Then we added students to buses, based on minimizing each unadded vertex's *diff* value (# of edges across new cut - # of edges added to set). This way, the solver would prioritize adding dense nodes that minimize cut friendships first. After greedily filling the buses, we accounted for rowdy groups by rearranging invalid vertices until all conflicts were fixed or the algorithm had run long enough that it cannot find anymore rearrangements.

The second algorithm is based on hMETIS[2], a hypergraph partitioning algorithm. A hypergraph is similar to a graph, except with hyper-edges rather than edges. Hyper-edges can contain more than two vertices, which allowed us to model rowdy groups. It is difficult to model rowdy groups in a graph because edges can only connect pairs of children, and do not accurately model rowdy groups of 3 or more children. However, utilizing hyper-edges, we were able to parse parameters.txt and form a low-weight hyper-edge for each rowdy group. We also parsed regular friendships from graph.gml into high weight hyper-edges. hMETIS minimizes the weight of hyper-edges that go across cuts, and can be used in both the k-way partitioning and recursive partitioning configurations. Because rowdy hyper-edges are lower weight, these hyper-edges are more likely to be cut by the partitioning algorithm. hMETIS does not take into account minimum or maximum partition size, so a separate helper function was written to help re-distribute children from buses overfilled by hMETIS. The helper function randomly picked however many excess children there were, and then distributed them among buses with space 100 times, and then picked the highest scoring result.

We ran our own solvers on our laptops and using an instructional machine. Each input has two outputs, one from each algorithm. We iterated through these outputs and returned whichever output scored higher.

[1] https://pdfs.semanticscholar.org/b981/195b306c39ef40939113bdc7e2bec5665191.pdf

[2] http://glaros.dtc.umn.edu/gkhome/metis/hmetis/overview

**Used libraries:**
networkx
os
from subprocess import call
from multiprocessing import Pool
random
sys
queue
from shutil import copy
from shutil import copyfile
from copy import deepcopy
pathlib import Path

**Standalone programs called from the solver:**
hmetis-1.5