Final Project Report

The first strategy was to find all of the cycles of length <= 5 in an instance, order them arbitrarily, and then solve the optimization problem with dynamic programming to select the subset of cycles which would maximize the penalty of the cycles chosen i.e. minimize the penalty suffered by the ones left over. The recurrence relation was:

DP(S) = Max{Score(n) + DP(S\{s_i}), DP(S\{s_n})} where S\{s_i} denotes the remaining subset of cycles to choose from after all those which conflicted with the nth one were removed.

The java cycle finding code was modified to skip an instance if it had not been solved in 60 seconds. The python cycle finding code was modified to cut any further recursion once the path length hit 5. The dynamic programming code was initially memoized recursion, and was later changed to be iterative with its own stack. We attempted to periodically trim the table of subproblems to deal with memory issues for large instances but still couldn't solve for these.

For the instances which this strategy was impractical for, we used Genetic Programming. The most difficult parts were initializing the population and creating a crossover function. For each individual, we ran depth-first search choosing random vertices at each step. If the DFS reached a dead end, it was deleted since this could not be in a cycle. We continued this process until all vertices had been either deleted or added as part of a cycle. For the crossover function, we picked two individuals at random, with higher probability based on its fitness score (number of adults in cycles and twice the number of children in cycles). Then, valid cycles were taken at random from each parent and added to the child. Once no more cycles could be added, more cycles were populated randomly, as in the initialization process. Bottlenecks in runtime were in population creation and crossover, which were parallelized and ran on an EC2 8-core processor. We found that running a population of thirty and thirty generations achieved optimal solutions when checked against our DP solutions, however due to computing costs, our final output ran a population of twenty and five generations.

Reference List:

Frank Meyer (2009) *Elementary Cycles Search* (Version 1.2) [Source Code]. Available at http://normalisiert.de/code/java/elementaryCycles.zip (Accessed 30 April 2016)

Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart, "Exploring network structure, dynamics, and function using NetworkX", in Proceedings of the 7th Python in Science Conference (SciPy2008), Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 11–15, Aug 2008