The Local Search Strategy

The program performs local search in a fairly straightforward way: after calculating a list of all conflicts, choose one at random, and flip both courses[1]. This works well, because *most* conflicts result when one course cannot be taken before another, but can be taken after, and thus flipping them will automatically solve the conflict.

However, there are cases where this is untrue: when two courses are in the same semester and cause a conflict, and for conflicts where a specific course has to be in a specific semester. In these cases, we use something akin to a random walk; if a conflict is between two courses that can't simply be flipped (as we know they would still be in conflict), we choose one randomly from the pair[2] and one randomly from the full course list, and assign this as the conflict instead. If the conflict is between a course and other external factors, we simply pair the course with a random one from the full course list.

In effect, we are making the assumption that most conflicts have an easily-known probable solution, and generating a random (much less probable solution) in all other cases.

If a solution exists, this will eventually find it, because given any single conflict without a certain solution, every possible single-flip solution will have a chance of being attempted, and when all single-flip solutions can be attempted, a cycle will never occur.

Other Design Decisions

A basic hashmap was used to store conflicts in Prereq. Given two courses, we should be able to determine if a rule exists for if/how those two courses conflict in O(1) time. Any given iteration runs in $O(n^2)$ time.

Custom rules can be specified using the PrereqRule interface to allow flexibility in implementation, and make fairly easy logic changes.

However, a simple List is used to record a semester schedule. The first, second, and third courses constitute the first semester, the fourth, fifth, and sixth constitute the second, and so-on. This makes swaps (and randomisation) much, much easier, but also critically improves the performance of things. However, it comes with the caveat that no semester before the last can be unfilled, and thus "dummy" course entries are used as a simple work-around.

[1] Exactly what algorithm this is is unclear. It is similar to a most-improving step in that it is choosing a step that is guaranteed to make an improvement, but it doesn't attempt to discern how many conflicts any given choice will improve. Thus, it is somewhat more like an any-conflict, except no variable is chosen at random from a conflict; the entire conflict is chosen and inverted.
[2] This part, at least, is much like an any-conflict algorithm.