Final Report

Computer Science 170 — Gabriel Perko-Engel — December 6, 2019

Before I go into *exactly* how my algorithm works on a macroscopic level, I'd like to start with its core component. Let $s = (p, p_0, d)$ be a possible solution to the Drive the TAs Home problem where p is the dropoff path that the car takes and d is the set of dropoff locations where the TAs are left before walking home. The list p_0 is an object I'm calling the "core path" that the vehicle takes. This is an ordered list of the locations the car stops at, the start/end location and the dropoff locations along the way. The fundamental process of my algorithm is the `Explore`(s) call which generates new solutions in the neighborhood of the solution s. It does this with three subroutines that I call `ExploreUp`(s), `ExploreDown`(s), and `ExploreAcross`(s). Of these, the `ExploreUp`(s) and `ExploreDown`(s) are the simplest. They simply add and remove one location from p_0 respectively. The `ExploreAcross`(s) subroutine picks a location within p_0 and swaps it out for some other one.

Let S be the set of all solutions generated so far by `Explore`(s) calls or otherwise. Initially, they only have their core path $p_0 = [\ell_0, \ell_1, \cdots, \ell_k]$ determined where ℓ_i is some location. I then generate the full path p by finding the shortest connecting path between all pairwise components ℓ_i, ℓ_{i+1}. I then find the dropoff locations d by picking each of the homes that a TA must be dropped off at and pairing them with the location that minimizes their walking distance. Finally, I update the core path to reflect the final dropoff locations.

Now that that's out of the way, we need to address how we determine what solutions get explored and where the initial solutions come from at all. My program starts my generating some number of "seed" solutions slected to try and make a representative sample of the solution space. This is done through a random process that first picks a number $0 < x \leq |H|$ of potential dropoff locations to build from. The locations are randomly selected from L and then constructed into a core path greedily by having the car always go to whatever next location has the shortest path from where it currently is. This is done multiple times for each chosen x and the xs are selected at numerous intervals across their valid range of values so that a representative number of starting solutions are chosen. In particular, I ensure that all of the 0 and 1 stop solutions are seeded as well as the full heuristic solution where Satish Rao drives each TA home individually.

All of the generated solutions are then stored as min heap sorted by their cost. After the initial seeded solutions are generated, my algorithm takes the m best solutions so far, explores them, and adds them to the heap. In this way only the best solutions will continue to have their neighborhoods explored and all the later ones will just be remembered so that if I encounter them again, I don't re-explore them. The way m is determined is inspired by simulated annealing such that $m = (a - b)e^{\epsilon/\tau} + b$ where ϵ is analagous to an energy level, τ to a temperature, a is the initial number of explored solutions, and b is the final number. I set τ by input size before running the algorithm and then have $\epsilon = |S|/\rho$ where ρ is the density of the graph as determined by `NetworkX`. In this way, τ sets the time scale of the explonential decay in the number of solutions explored in pursuit of the final. The algorithm ends either when all of the solutions in the neighborhood of the m best ones are worse than the m best ones or when some predetermined time t has elapsed. The task then is just to tweak a, b, τ, and all the additional random parameters that determine how many solutions we generate of what type on each given `Explore`(s) call.