The Single Fleet Problem

KAPT Team

June 30, 2023

Inputs

We are given a strongly connected weighted di-graph $G = (V, E)$ with subsets $A, S, F \subset V$ representing the assembly points, sinks, and starting locations of vehicles, respectively, where the weight of edge (u, v) represents the time it takes for a vehicle in the fleet to travel from u to v. The nodes are 1-indexed, with A being indexed as the first $|A|$ nodes, then S as the next $|S|$, then the rest. As well, we are given two functions $n : A \to \mathbb{Z}_+$ and $r : A \to \mathbb{R}_+$ such that $n(a)$ is the total amount of people that will eventually gather at a and $r(a)$ is the rate of assembly of such people. Note that $r(a)$ is to be thought of as the rate of an exponential random variable modeling inter-arrival times between people at assembly point a. Finally, we are given two integers K and C denoting the number of vehicles to route, each with capacity C. Each $a \in A$ is associated the value $\lceil f(a)/C \rceil$, which is the number of times it must be visited. Additionally, to each $a \in A$ we associate the following set of intervals: Let μ be the expectation of a gamma random variable with parameters $n = f(a)$ and $\lambda = r(a)$ with σ as the standard deviation of the same random variable. Then we associate the set of intervals $\{[k\mu - \sigma, k\mu + \sigma] : k = 1, ..., \lceil f(a)/C \rceil\}$ to a.

Problem Statement

The goal of the problem is two-fold. First, we need to choose a vertex $v \in V$ to act as the depot for vehicles to gather at the beginning of the NEO. Secondly, we need to find a set of at most K walks that minimizes the max time of these walks. If $(v_0, v_1, ..., v_n)$ is a walk in the output, then the set of walks must obey the following constraints:

1. $v_0 = v_n = d$, with n odd for all walks.
2. $v_k = a$ for some $a \in A$ for all k odd such that $k < n$ for all walks.
3. $v_k = s$ for some $s \in S$ for all k even such that $k < n$ for all walks.
4. For each $a \in A$, a is included on these walks exactly $\lceil f(a)/C \rceil$ times.
5. For each $a \in A$, the k^{th} time a is visited is minimally late with respect to its corresponding time interval $[k\mu - \sigma, k\mu + \sigma]$.

Algorithm

The goal of the algorithm is to create a graph object that is an instance of the vehicle routing problem with time windows (VRPTW), solve that problem, then reconstruct the solution to our problem in a way that automatically satisfies the constraint outlined in the problem statement. We begin by running Johnson's algorithm on G to obtain an APSP matrix, which will be useful in multiple steps of the algorithm. We begin by choosing the depot, which is chosen as the node satisfying

$$\arg\min_{v \in V} \left(\sum_{a \in A} \delta(v, a) + \sum_{u \in F} \delta(u, v) \right)$$

where $\delta(u, v)$ represents the shortest distance of a path from u to v, which is easily obtained from the APSP matrix. We denote the chosen depot as d. Now, we tackle the harder problem of routing the vehicles.

We construct a complete graph such that a routing through it will automatically obey constraints 1 through 3. Let $V_a = A \bigcup \{d\}$. Define a new graph $G_a = (V_a, E_a)$ where $E_a = \emptyset$. For each $a \in V_a \setminus \{d\}$, add the edges (d, a) and (a, d) to E, with weights $w((d, a)) = \delta(d, a)$ and $w((a, d)) = \min\limits_{s \in S} \delta(a, s)$. Then, for $a_1, a_2 \in V_a \setminus \{d\}$, add the edge (a_1, a_2) with $w((a_1, a_2)) = \min\limits_{s \in S} (\delta(a_1, s) + \delta(s, a_2))$. Note that the argument of the previous function is kept track of and will be used later in the algorithm. Now, G_a is a $|V_a|$ - complete graph. We represent this graph as an adjacency matrix M with index 0 denoting the depot and the remaining $|A|$ indices representing the assembly points consistent with the ordering in the problem set-up, where the entry $M_{ij} = w((i, j))$.

We now modify the resulting graph further such that a routing automatically obeys the remaining constraints 4 and 5. To do so, we increase the complexity of the graph by duplicating each $a \in V_a \setminus \{d\}$ but maintain the completeness of the graph in a way that any valid routing still makes sense for the problem. Observe that the new graph we create will have $1 + \sum_{a \in A} \lceil f(a)/C \rceil$ nodes. We will create the new graph through a series of matrix manipulations to M to create a new adjacency matrix. Consider the i^{th} row in M. The transformation we apply to the row is inserting $\lceil f(j)/C \rceil - 1$ copies of M_{ij} directly after column j for all columns j. Note that entries are only once-duplicated, meaning we keep track of the original columns and only duplicate those entries in this manner. Further note that this creates a row vector of the desired size. Then, to construct our new graph, we begin by transforming each row as detailed above to obtain a new set of row vectors $\{M'_{i,*}\}$. Finally, to construct our final matrix, we begin by adding the depot row $M'_{0,*}$. Then, for each $i = 1, ..., |A|$, we add $\lceil f(i)/C \rceil$ copies of $M'_{i,*}$ below the most recently added row. Let this new complete graph be denoted G_a^*. To complete the instance of the VRPTW, we need to assign time intervals to each column of G_a^*. To do so, first assign the interval $[0, 1]$ to the depot. Then, assume the first assembly point was duplicated n times. This node has associated to it a set of n time intervals, so we just assign these time intervals to columns 1 through n. Then, we do the same with the second assembly point beginning at column $n + 1$. We repeat this until all intervals have been assigned.

At this point, we have an instance of VRPTW, so we solve this problem to obtain a routing on G_a^*. To do so, we use a hybrid genetic search algorithm as outlined by python's vehicle routing problem library. Although the problem instance may not contain a feasible solution so that every time interval is obeyed, it does its best to minimize lateness. Thus, we run this algorithm to obtain a routing on G_a^*. To complete the algorithm, we only need to reconstruct the complete routes by including the sinks that we chose in the beginning between each assembly point. But, because we kept track of this, it is easy to insert a new between each assembly point and connect it with edges of weights given by the APSP matrix.