The quadrotor route finding problem is a variant of a pickup-and-delivery problem (PDP). It is expressed here as a linear program. We will make use of the following sets:

* is the set of all ground vehicles (base stations for the quadcopters).
* is the set of all quadcopters.
* is the set of all items available in the request system.
* is the set of requests that must be served.
* is the collection of all important locations
* is the set of directed edges in the graph on

For , we will denote the distance between the endpoints of the edge by . Sometimes, we will refer to an edge by its ordered pair.

The variables of the linear program are:

* Binary variables with value 1 if quadcopter h traverses edge e and 0 otherwise
* Quantity of item s carried along edge e (integer)
* Quantity representing the state of the battery of quadcopter h after traversing edge e
* Abstract quantity roughly quantifying the number of legs remaining in the trip for quadcopter h after traversing edge e.
* Time required for all quadcopters to complete their routes

The objective function is then easily expressed as,

There are a large number of constraints, so we will begin with the bound constraints:

* (Eliminate self-loops)
* , where quadcopter starts at . (No one can fly to the starting location of a quadcopter)
* , where quadcopter starts at with initial charge, . (Remaining battery after the first leg is bounded by the initial charge less the battery consumed on the first leg)
* (Remaining battery after departing a station is bounded by a full charge, less the battery consumed in departing the station)
* , where quadcopter starts at and . (No quadcopter may depart from another quadcopter’s starting location)

The remaining equality and inequality constraints are broken up into a number of types.

* Type 0 constraints. Each quadcopter can only leave its starting location in one direction.
* Type 1 constraints. The quantity of each item carried away from each quadcopter’s starting location is equal to the quantity of that item that particular quadcopter is currently carrying.
* Type 2 constraints. No route is flown twice.
* Type 3 constraints. Every request is serviced.

Here we use the convention that a delivery is a positive number of items and a pickup is negative. Also, corresponds to the location of request .

* Type 4 constraints. Every request is visited once.
* Type 5 constraints. Cargos must respect the carrying capacity of the quadcopters.
* Type 6 constraints. A quadcopter moves to a request if and only if it moves away.
* Type 7 constraints. A quadcopter arrives at each station at least as many times as it leaves.
* Type 8 constraints. Battery is only consumed on legs that are travelled.
* Type 9 constraints. The amount of battery left after leaving a request is equal to the amount of battery when arriving at that request less the amount of battery consumed in leaving the request.

where is the location of request .

* Type 10 constraints. The number of legs remaining in a route only makes sense for edges in the route.

Here, was chosen as an upper bound for the number of legs in a minimal route, effectively leaving unbounded wherever is positive.

* Type 11 constraints. The number of legs remaining after departing a node (other than a starting node) is equal to the number of legs remaining upon arriving at that node less the number of departures from that node.

where is the location of request . Although non-obvious, this is valid even if the quadcopter visits a node multiple times. The first arrival and last departure have meaningful values although any intermediate circuits may have shifted values.

* Type 12 constraints. The total time taken is at least as long as the time taken by each quadcopter. (this is how we fake a max() constraint)

For the actual implementation, variables are arranged into blocks, corresponding to the edge index. Within each block, the nodes are arranged in the order, request locations, quadcopter start locations, then ground vehicle locations. The variables are placed in column-major order within each block. The blocks are then arranged in order with x values first, followed by q’s, then z’s, then t’s.

As an example, suppose we are dealing with a single quadcopter, single request, single item, and single ground vehicle. The request will then be at location 1, the quadcopter at location 2, and the ground vehicle at location 3. The variables in the linear program will be arranged in the order,

,T

We can also make a number of preliminary cuts. These are inequalities that follow from the integrality constraints, but not from the linear constraints. Including these cuts isn’t necessary, but they do constrain the linear relaxation, which can reduce solution time.

* Additional bounds. The legs departing from a quadcopter’s starting location can be constrained based on cargo. If the end of the leg is a delivery and the quadcopter’s cargo does not include the goods to be delivered, then that leg cannot be taken. Similarly, if the end of the leg is a pickup and there is insufficient room remaining in the quadcopter’s capacity, the leg cannot be taken.
* Type 13 constraints. All quadcopters must move. If a quadcopter is already at a vehicle we want it to wait, but this can be satisfied by moving the 0 distance to the vehicle.