Approaches to Two-Stage

Over the week I created the following methodologies for solving two-stage optimization.

1. Run only jobs first to get an absolute jobs minimum. Then run coverage.
2. Run column generation on jobs and coverage simultaneously. That should be enough.
3. Run column generation on jobs and coverage simultaneously, until one of them stops producing routes. Then, run colgen on the one that didn't stop until the end.
4. Produce a set of feasible routes for both coverage and jobs. Then, perform column generation on one of them until the end; then, the other one until the end; and so on, alternating, until both of them fail to do any better.
5. Run column generation on jobs and coverage simultaneously, for however long you'd like. If it has the repeated xi_it issue, or during any time in the process, feel free to stop. Then, colgen on one and then the other until they both stop doing better.

Among these we see many distinct column generation schemes and subproblems, and the five methods above are a combination of these algorithms.

Jobs alone*

Min $\sum_q C^q z^q$

st $\sum_q u_i^q z^q = 1 \ \forall i \ (\pi_i)$

st $\sum_q z^q \le B \ (\rho)$

Dual

Max $-B\rho + \sum_i \pi_i$

st $-\rho + \sum_i u_i^q \pi_i \le C^q \ \forall q$

Subproblem

Min $C^q + \rho - \sum_i u_i^q \pi_i$

Jobs and Coverage

Min $\sum_q C^q z^q + \sum_p C^p x^p$

st $\sum_q u_i^q z^q = 1 \ \forall i \ (\pi_i)$

st $\sum_q z^q \le B \ (\rho)$

st $\sum_p x^p \le V \ (\beta)$

st $\sum_p \sum_j L_{ji} y_{jt}^p x^p \ge \sum_q z^q \delta_{it}^q \ \forall i, t \ (\xi_{it})$

Dual

Max $\sum_i \pi_i - B\rho - V\beta$

st $-\sum_i \sum_t \delta_{it}^q \xi_{it} - \rho + \sum_i u_i^q \pi_i \le C^q \ \forall q$

st $-\beta + \sum_i \sum_t \left(\sum_j L_{ji} y_{jt}^p \right) \xi_{it} \le C^p \ \forall p$

Subproblem

Min $C^q + \rho - \sum_i u_i^q \pi_i + \sum_i \sum_t \delta_{it}^q \xi_{it}$

Min $C^p + \beta - \sum_i \sum_t \left(\sum_j L_{ji} y_{jt}^p \right) \xi_{it}$

Coverage alone

Min $\sum_p C^p x^p$

st $\sum_p \sum_j L_{ji} y_{jt}^p x^p \ge W_{it} \ \forall i, t \ (\xi_{it})$

st $\sum_p x^p \le V \ (\beta)$

Dual

Max $-V\beta + \sum_i \sum_t W_{it} \xi_{it}$

st $-\beta + \sum_i \sum_t \left(\sum_j L_{ji} y_{jt}^p \right) \xi_{it} \le C^p \ \forall p$

Subproblem

Min $C^p + \beta - \sum_i \sum_t \left(\sum_j L_{ji} y_{jt}^p \right) \xi_{it}$

* In the above five methods, often we will run column generation given a coverage setup. As you will see later, this formulation also works for "jobs alone given coverage"—all we have to modify is the time windows and I will prove its validity.

This means we have to write 3 different column generation loops and 3 different shortest paths. 3 column generations, because either we handle jobs, coverage, or both. I will write these loops as functions with inputs and outputs. 3 shortest paths, because there is one for coverage (notice how, in both the coverage AND coverage & jobs, they happen to have an identical subproblem taking in identical parameters which may differ from the outside, but not within the function), and two for jobs (one for jobs alone, and one for jobs and coverage, which admits an extra two parameters in delta and xi).

I will write all six of them as their own independent functions. In other words, they are standalone. This canonizes the components and makes it easier for us if we want to make any modifications. It also proves the trustworthiness of each component alone.

Subproblem Algorithms

<u>Jobs Alone</u>: Perhaps the easiest to write.

Input:
n_jobs
job_dists   job_travel_times   windows   loads
rho   pi_i

Output:
node_sequences, time_sequences, reduced_costs

<u>Jobs in Conjunction with Coverage</u>: A bit modified, but not too bad.

Input:
n_jobs
job_dists   job_travel_times   windows   loads
rho   pi_i   xi_it

Output:
node_sequences, time_sequences, reduced_costs

<u>Coverage</u>: The most complicated. We will use helper functions.

Input:
m_cov   n_jobs
cov_dists   cov_travel_times
beta   xi_it

Output:
node_sequences, time_sequences, reduced_costs

Helper Functions:
when_to_leave: The times when you are allowed to leave the spot.
when_to_enter: The times when you are allowed to enter the spot.
all_jobs_ended: If all jobs have finished at a spot (so you never enter it).
neighborhood: Given a spot, tell me which jobs are nearby.
close_covs: Given a job, tell me which spots are nearby.

Column Generation Loops

Jobs alone without with coverage:*

Input:
n_jobs B
job_routes
job_dists job_travel_times windows loads

Output:
last_z, job_routes, job_objective

Helper Functions:
compute_route_cost
compute_u
compute_delta
prune_time_windows

Loop Shape:
1. Initialize model
2. Compute parameters u, delta, y, costs for the first time. Initialize Q (number of routes).
3. Prune time windows to fit with coverage availability.
4. Enter while loop
    1. Fill model
    2. Optimize model and set job_objective and last_z
    3. Extract dual variables rho and pi_i (hand-written or automatic)
    4. Enter jobs-alone shortest paths algorithm.
    5. Extract nodes, times, and reduced costs.
    6. Calculate best route to add.
    7. If best route is empty, leave loop.
    8. Append best route to job_routes.
    9. Re-calculate u, delta, costs, Q.
5. Exit while loop
6. Return last_z, job_objective, job_routes (last_z length is shortest).

* I claim that the orange fix is the only thing that differentiates jobs without considering coverage availability and jobs with considering coverage availability. This is because we know mathematically that coverage vehicles are not allowed to leave their spot while a job is going on anywhere in their neighborhood (and I mathematically proved this). Therefore, if coverage is to falter, it will do so at the ends of a time window, not during

the middle. As a result, all I need to do to make job routes compatible with coverage limitations is to snip the time windows as necessary and use these 'new' time windows throughout the problem. Remember that coverage is stationary/a constant in this formulation. Also, recall that coverage solutions are only formed with job results considered—we never solve coverage vehicles in isolation, or the objective will be 0.

Jobs and Coverage:
* For this one, put an iterations limiter because it is prone to running forever! *

Input:

n_jobs	m_cov	B	V
job_routes	cov_routes		
job_dists	job_travel_times	windows	loads
cov_dists	cov_travel_times		
num_iters			

Output:
last_z, last_x, job_routes, cov_routes, full_objective

Helper Functions:
compute_route_cost
compute_u
compute_delta
compute_L
when_to_leave
when_to_enter
all_jobs_ended
neighborhood
close_covs

Loop Shape:
1. Initialize model
2. Compute parameters u, delta, L, job costs, cov costs. Initialize Q and P.
3. (Should we use time windows or work windows based off the last iteration of the jobs routes? See "Questions Yet Unexplored".)
4. Enter while loop (or: for iteration in 1:num_iters)
    1. Fill model
    2. Optimize model and set full_objective, last_z, last_x
    3. Extract dual variables rho, beta, pi_i, xi_it (hand-written or automatic)
    4. Compute new job route from jobs-and-coverage shortest paths and new coverage route from coverage shortest paths.
    5. Extract nodes, times, and reduced costs for both.
    6. Calculate best routes to add.
    7. If best route for one or both is empty (see "Questions Yet Unexplored"), exit loop.
    8. Append best routes as necessary.
    9. Re-calculate Q, P, u, delta, L, job costs, cov costs.
5. Exit while loop
6. Return last_z, last_x, job_routes, cov_routes, full_objective.

Coverage alone:

Input:

m_cov	n_jobs	V
cov_routes	job_routes	windows (flexible: work windows? time windows?)
cov_dists	cov_travel_times	

Output:
last_x, cov_routes, cov_objective

Helper functions:
compute_route_cost
compute_W
compute_L
compute_y
when_to_leave
when_to_enter
all_jobs_ended
neighborhood
close_covs

Loop Shape:
1. Initialize model
2. Compute parameters y, L, W. Initialize Q and P.
3. **Create windows that reflect when job work is done, NOT time windows!**
4. Enter while loop
    1. Fill model
    2. Optimize model and set cov_objective, last_x
    3. Extract dual variables beta, xi_it
    4. Compute new coverage route from algorithm.
    5. Extract new path and see if you add anything.
    6. Re-calculate y, L, W, Q, P.
5. Exit while loop
6. Return last_x, cov_routes, cov_objective.

We need job_routes so we can calculate W_it.

Helper Functions
1. compute_route_cost: how much a route costs.
2. compute_u: parameter u_i^q from job routes.
3. compute_delta: parameter delta_i^q from job routes.
4. compute_W: parameter W_it from job routes.
5. compute_L: parameter L_ji from job locations and coverage locations.
6. compute_y: parameter y_jt^p from coverage routes.
7. when_to_enter: based on provided windows, return times of arrival.
8. when_to_leave: based on provided windows, return exit times.
9. all_jobs_ended: when all jobs are finished based off provided windows.

10.neighborhood: given a spot, list all nodes in the vicinity.
11.close_covs: given a node, list all spots in the vicinity.
12.prune_time_windows: given coverage routes, cuts time windows as appropriate.
13.compute_work_windows: given job routes, produces work windows as fit.
14.route_to_nodes: given route, list nodes
15.route_to_times: given route, list times
16.nodes_and_times_to_route: given nodes and times, create route

How Each of the Five Methods Works

1. Jobs-alone loop. Then Coverage-alone loop.
2. Jobs-with-coverage loop.
3. Jobs-with-coverage loop. Then one-that-didn't-stop loop.
4. Jobs-alone loop, then coverage-alone loop...
5. Basically 2 first, then 4.

Canonical Representation for Time

For node and time sequences, we do the following:
start-depot, L1, L2, ..., end_depot
0, T1A, T1D, T2A, T2D, ..., end_time

where T1A represents the time of arrival to node L1, and T1D the time of departure from node L2.

In other words, work or coverage is performed between Ti/jA-Ti/jD, with travel between Ti/jD-T(i+1)/(j+1)A.

Observation

Job routes are like vehicle routing with time windows.
Coverage routes are like vehicle routing with multiple time windows and the possibility to return back.

Questions Yet Unexplored

1. When generating coverage from jobs in the double loop, should we base restrictions for when coverage routes can enter a location off the current solution for jobs (i.e. work windows), or the time windows (which are wider)? As it is currently configured, we can do either.
2. Should we exit the joint loop if one or both of them stops producing routes with non-zero reduced cost?
3. Will >= versus == for that constraint matter anymore?
4. How do we deal with the pathological example I mentioned back about the possible downsides of working early?