Real world newspaper distribution

Morten Smedsrud, SINTEF ICT

The newspaper delivery vehicle routing problem

- Two step process, first step is from the printing press to drop points and second step from drop points to subscribers (which is the part we work on)
- Some routes are driven (with private cars), others are walked by pedestrians
- Car routes are open with no return to drop point, pedestrian routes must return to drop point with trolleys (opposite in Sweden)
- No practical capacity constraints, time however is constrained, all routes typically must be served within 2 hours
- Relatively large number of customers in given problem instances (1000-10000+)
- Sometimes minimum number of routes must be determined, other times number of routes are given

The NDVRP part 2

- One main goal is to minimize total route duration
- Routes need to be balanced time wise, at least within ~20%
- Routes need to be "visually appealing", i.e.: each route must be compact and there can be no overlap between routes
- Asymmetric travel cost / distances / times
- The travel time model must take into account whether meandering ("zigzag") is allowed or not

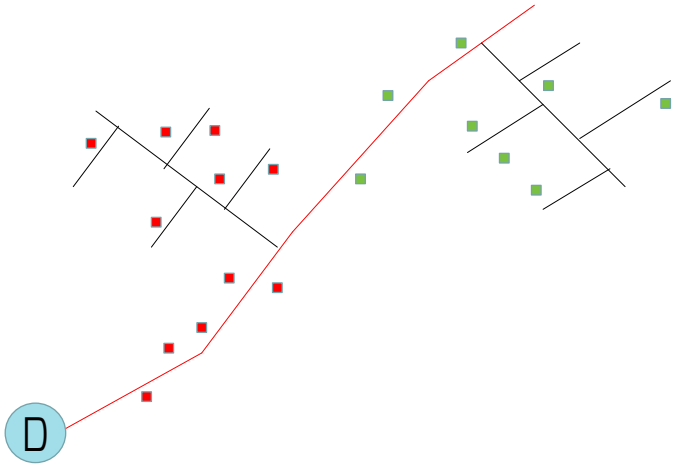
First attempt, solving with Spider

- Reducing number of orders by aggregating orders belonging to same road link into arc locations and solving resulting arc routing problem
- Introduce objective to measure how clustered solution, i.e. penalises overlap between routes
- Introduce objective to measure how balanced the routes are, penalises variable duration of routes
- Introduce new restrictions into Spider Topology module (zigzag).

Problems with the objective approach

- Letting the cluster cost be high may result in solution with tours following the natural clusters in a problem. But they are often badly balanced
- If balancing objective dominates the objective function, the optimizer might balance routes by doing stupid things like driving back and forth inside the shorter tours to even them out.
- Seems impossible to balance the objectives

Topology problems



- Tours from a depot out to clusters of orders often goes along central roads
- Orders along this central road may be serviced by any route passing through without any extra travel costs.
- Makes the optimizer very proficient at creating overlapping routes and destroying existing clusters when running without cluster objective.
- Not allowing zigzag delivery often creates further complications, when travel distance to an order suddenly can vary significantly depending on what side of the road you come from

New approach, initial solver that makes balanced clusters

- Orders are aggregated as usual
- Orders are then clustered in a initial solver
- Initial solver also balances these clusters
- We adapt a known clustering algorithm like k-means to build clusters modify it to balance tours
- Once clusters are built, we build tours from the resulting clusters in Spider
- Spider then optimizes the resulting tours individually, no inter tour optimization (which can destroy the clustering)

K-means algorithm

1. Select a random location as first centre
2. For each additional cluster pick the location furthest away from previously selected centres and make it centre for this cluster,
3.
4. Assign each location to the cluster who has the closest centre, i.e. cluster can be expressed as:

$$\{ \quad \| \quad \| \quad \| \quad \| \quad \{ \quad \} \}$$

5. Update each centre to new centroid:

$$\overline{\| \quad \|}$$

6. Increase t by 1 and go back to step 4 until $\{ \quad \}$

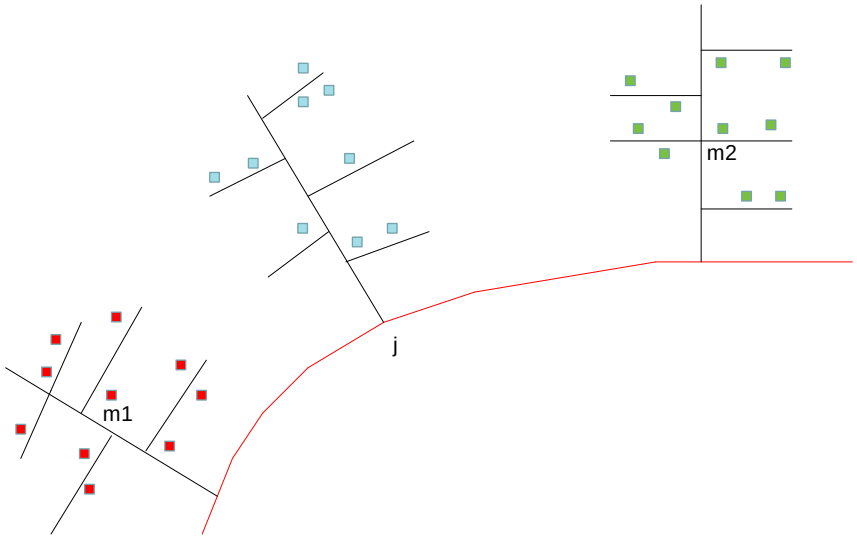
Adaptations to our problem

- No centroids in a road topology (could be in a lake or something), so picks closest location in Euclidian distance when updating centres in clusters
- Use non modified K-Means to generate a (unbalanced) starting solution
- In order to be able to balance routes we need to know how big a cluster is, done with a simple and quick TSP solution after locations have been assigned to clusters
- Balance routes by introducing an offset when assigning locations to cluster:

$$\left\{ \quad \| \quad \| \quad \| \quad \| \quad \{ \quad \} \right\}$$

- For each iteration we either balance routes by growing or shrinking the most unbalanced cluster by modifying the offset for that cluster, or when balance has been stable for a while we update centres like in K-Means
- Terminate on time or when all routes are balanced within certain tolerance and estimated time for all clusters are feasible

Road topology balancing problem



- Algorithm doesn't manage to split middle cluster when trying to service above orders with 2 clusters
- Travellers travel through junction j regardless of whether we come from centre or , so all orders in middle cluster get assigned to either centre depending on value of offsets and
- Solution: use a linear sum of travel distance and Euclidian distance (emphasis on travel distance)

Minimizing number of routes

1. Start with an estimated number of tours , set to a lower bound for how many routes needed, i.e. total service time of all orders versus available time on tours
2. Use the previously described approach to cluster the plan number of tours
3. If the plan can be feasibly clustered with this number of routes, then calculate slack to estimate how many tours can be saved and subtract this number (or minimum 1) from and go back to step 2
4. If the plan with tours is not feasible, set to new estimate (increase with minimum 1) for how many tours are needed and go back to step 2
5. Terminate at step 3 or 4 when tours are infeasible but tours are feasible

Parallelization

- The route minimization procedure can easily be multithreaded trying to check if several different number of routes are feasible simultaneously
- Use multithreading to cluster different plans independently simultaneously, and return first balanced solution found by any thread. Small benefit as each thread will need some time to stabilise before starting to return balanced solutions.
- Use multithreading to cluster different plans independently simultaneously,, but running a given amount of time and pick best solution. This approach need some way to evaluate the solutions (for instance how well they are clustered, how balanced routes are and / or total duration). Improves quality but not response time.
- Parallelize entire algorithm. Real bottleneck is the TSP for estimating cluster sizes, possible big benefits if this can be run fast on a GPU. Expensive but biggest potential for speed up.