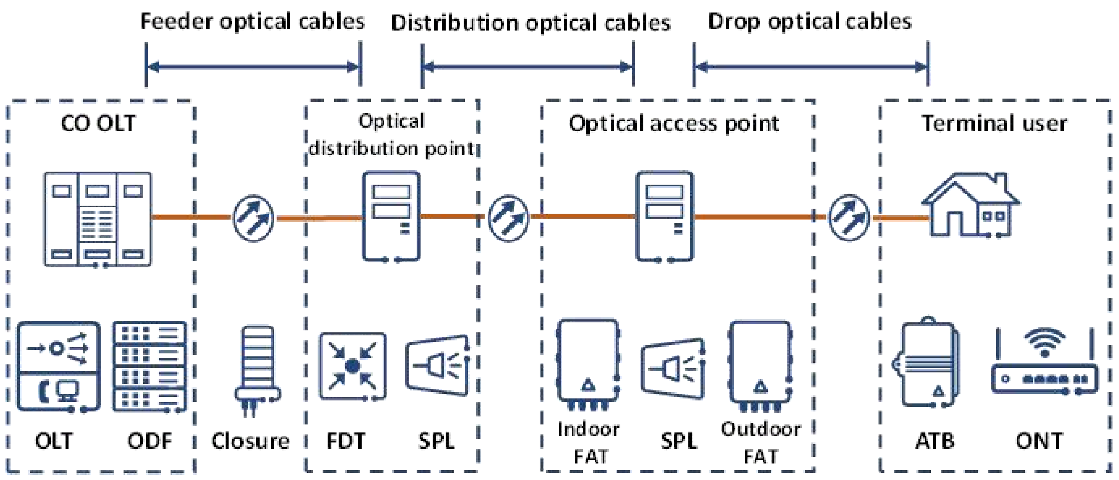
Brief Description of the Huawei ODN planning Problem

Zhongxuan Li

This is an optimization project which I took principle responsibility in 2019 - 2020. ODN (optical distribution network) is the peripheral of the optical access network. Unlike the backbone transportation network, ODN has versatile topologies. However in this article, we focus on two major segment of ODN i.e. star topology of the drop optical cable network, and daisy chain topology of the distribution optical cable nework. The ODN planning problem aims in reducing the total cost of cables, right-of-way and number of ports (e.g. boxes, OLT stations). In the following paragraphs, we formulate the optimization problem in mathematical language and discuss them in detail.



ODN Drop Optical Cables (Layer one) Problem

ODN drop optical problem can be transformed into the a Single Source Capacitated Facility Location Problem (SSCFL), where each FAT (Fiber Access Terminal) is equivalent to a 'facility'. Users are connected to FAT in a star topology.

The drop layer problem is formulated as follows,

- The set of **demands** D (users or homes or points).
- The set of **access nodes** V (places, where we can install FAT).
- The set of **possible arcs** A. Every arc $a \in A$ is a tuple (v, d), where $v \in V$, $d \in D$. An arc (v, d) is present in A, only if the distance between v and d is not greater than $len_{\max} = 200$ (meters). So, the set A have to be generated in the preprocessing stage.
- The **length** $c_a \in \mathbb{Z}^+$ for every $a \in A$.
- The **parameter** $degree_{\max} = 16$ — the maximal number of demands that can be connected to one FAT.

Let $x_v \in \mathbb{Z}^+$ be the number of FATs that will be installed in an access node v, $y_a = 1$ for $a \in B$ and $y_a = 0$ otherwise. So, $\boldsymbol{y} = (y_a) \in \{0,1\}^A$. Thus, we have to:

$$\text{minimize} \quad M \sum_{v \in V} x_v + \sum_{a \in A} c_a y_a$$

$$\text{subject to:}$$

$$\sum_{v \in V} y_{v,d} = 1, \quad \forall d \in D,$$

$$\sum_{d \in D} y_{v,d} \le x_v \cdot degree_{\max}, \quad \forall v \in V.$$

M is a big constant. For example, $M = degree_{\max} \cdot c_{\max}$, where $c_{\max} = \max_{a \in A} c_a$.

Furthermore, we modified the problem to an “intersection free” mood. Because the possible arcs set A is obtained by Dijkstra’s algorithm, we could therefore know which two possible arcs have common nodes, shared line segments, or continuous plane. These are known as point, line, and plane intersections.

The following two charts are examples of “plane” intersection:

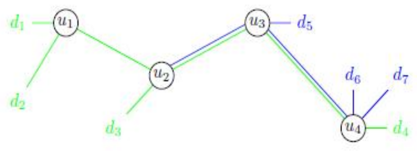


Figure 1: Green and blue trees can be splitted

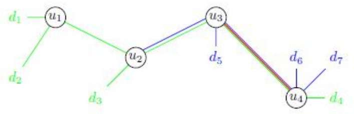
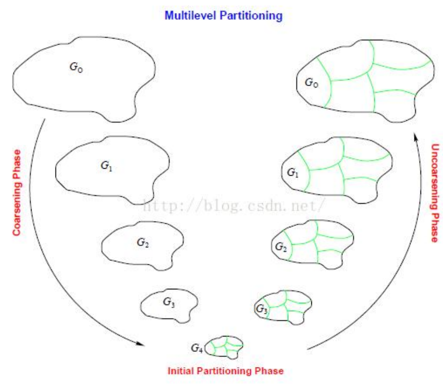


Figure 2: Green and blue trees can’t be splitted. Red line is a place of intertwining

Proving we have acquired the intersection relationship between any two possible arcs. We can add logical OR constraints to those two arcs, such that $y_1 + y_2 <= 1$

In order to give a speedup to the model, we adopt a sequential graph partition framework, which applies the famous multilevel graph partition algorithm (written in the METIS software). For each iteration, we only solves a part of the complete roadmap, until all users have been connected.



In addition, a network-flow based model has been proposed to solve the “point intersection” SSCFL problem. The model is described as follows,

Let $S = U \cup W$ be a set of “non-demands”. Let $A \subseteq S \times S$ contains two pairs (s_1, s_2) and (s_2, s_1) for every edge $\{s_1, s_2\} \in E$, $s_1, s_2 \in S$. Let $c(s)$, $s \in S$, be the number of edges $\{s, d\} \in E$, $d \in D$. For every access $u \in U$, we have a variable $x(u) \in \mathbb{Z}_+$ (the number of FATs). For every arc $a \in A$, we have a variable $y(a) \in \mathbb{Z}_+$ (the number of paths containing a). Let M be a very large constant.

Thus, we have to

$$\text{minimize} \quad M \sum_{u \in U} x(u) + \sum_{a \in A} \text{len}(a) y(a), \tag{1}$$

subject to

$$\sum_{(s,w) \in A} y(s,w) - \sum_{(w,s) \in A} y(w,s) = c(w) \quad \forall w \in W, \tag{2}$$

$$\sum_{(u,s) \in A} y(u,s) - \sum_{(s,u) \in A} y(s,u) + c(u) \leq x(u)\,\mathrm{M}_{\deg} \quad \forall u \in U, \tag{3}$$

$$x(u) \in \mathbb{Z}_+ \quad \forall u \in U, \tag{4}$$

$$y(a) \in \mathbb{Z}_+ \quad \forall a \in A. \tag{5}$$

The drawback of this model is: it does not include distance constraints. Even though we can add on distance constraints by MTZ constraints (Miller, C. E.; Tucker, E. W.; Zemlin, R. A. (1960). "Integer Programming Formulations and Travelling Salesman Problems". J. ACM. 7: 326–329. doi). But the model would become slow. Furthermore, on complex roadmap, this “flow model” takes a lot of time to reach relative optimal gap (in branch and bound).

Objectives:

Problem size of Layer one ranges from 500 users to 5000 users, whereas the roadmap can have up to 20,000 edges and 9,000 nodes. Of course, the possible arcs obtained by Dijkstra's algorithm have an even larger scale.

Variables and constraints:
FAT location are to be solved;
Users' location and number are pre-defined;
Maximum cable length constraints must be met;
Capacity of the hub box must be met;

Problem scale: 500-5000 users to be connected

We aim to minimize the number of FATs used (primary goal), and the total routing distance (second goal), with intersection free, and under distance, capacity constraints.

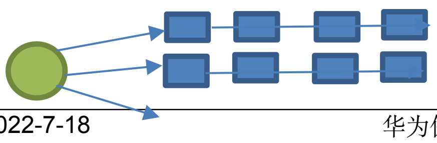
Speed of the algorithm is also a very important factor taken into consideration.

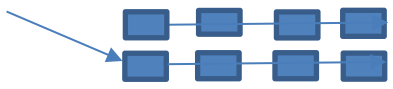
Ways to improve:
1. Better (Lean) model of formulations
2. Reduce the number of "intersection" constraints
3. A better graph partition method, other than METIS
4. Better post-processing methods, merging two under-volume clusters, e.g. merge three clusters into two, five into four, etc. Very similar to Jewel of Atlantis.

ODN Pre-Connection (Layer two) Problem

The pre-connection problem targets at FDT -> FAT of an ODN network. Conventionally, 4 FATs are connected in a chain manner, whereas 4 chains are joint together (at a starting node, FDT), forming a complete pre-connection topology.

The picture below illustrates what is a pre-connection topology, where 4 chains are jointly connected at the FDT depot (green node).

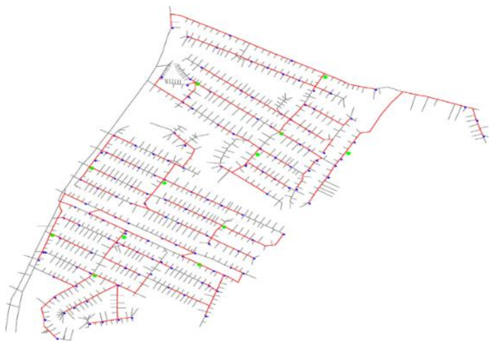




The layer two problem is very similar to The Multiple Depot, Multiple Traveling Salesmen Facility-Location Problem. (The Multiple Depot, Multiple Traveling Salesmen Facility-Location Problem: Vehicle Range, Service Frequency, and Heuristic Implementations - YuPo CHAN)
One of the modelling technique is to mimic the problem as a MDMTSFLP. Given the very large problem scale, this approach is considered to be time consuming. In addition, chains belongs to different FDTs must not intersect.

Hence, we applied a two-stage manner approach. Using a similar flow model as in the layer one task, the roadmap is first split into several node-disjoint trees. Then we solve the routing problem inside each tree by the capacitated vehicle routing model (CVRP).

The chart below illustrates the preliminary result after the splitting stage, after which CVRP is applied on each depots, inside on selected tree (in red).



Furthermore, we care of route repetition rate, which is the duplicate routes/total routing distance traversed by distinct chains origins from FDT.

Variables and constraints:
FDT location are to be solved;
FAT numbers/locations are pre-defined;
Maximum cable length constraints must be met;
Capacity of the hub box must be met;

Problem scale: 30 – 300 FAT to be connected
We aim to minimize both FDT number and total routing distance with an emphasis on FDT number.
Time performance is notably important.

Ways to improve:
1. A fast, effective heuristic methods in replacement of the current integer programming

model. Capacitated minimum spanning tree sounds promising, but we cares more about reducing the number of FDT, not total distance. Therefore, trees are intrinsically minimum.

2. Solve the whole pre-connection problem as one “Location Routing Problem”, rather than two stages.

Computing platform and time:

The current algorithm runs on laptops, normally Intel i9 with 32G of RAM
Time Limit: The current algorithm solves 5000 demand users within 10mins.

For longer term research purpose, the algorithm may be migrated to a server (Intel Xeon, 128GB) .Time performance should be less than 10 minutes.