Passive Optical Network Planning

1 Introduction

Passive Optical Networks (PONs) are systems that use point-to-multipoint fiber to the premises in which unpowered optical splitters are used to enable a single optical fiber to serve multiple endpoints. Designing a cost-effective and efficient PON is critical to ensure optimal performance and investment returns.

2 Problem Statement

Given a distribution network, the objective is to plan the optimal placement of splitters and the routing of fibers in such a way that the total network cost is minimized.

2.1 Specifications

- The central office is represented by squares and serves as the main connection hub.
- Optical Network Units (ONUs) are endpoints represented by circles (type).
- Splitters have a fixed capacity of 32 and are represented by triangles (type).
- The network graph is provided, which outlines the potential placement locations for splitters and the links (edges) connecting them.
- Each edge (fiber) has a cost per meter.
- Digging trenches has an associated cost.
- It is possible to lay multiple fibers in a single trench, termed as fiber duct sharing.
- Each splitter has an associated cost.

2.2 Decision Variables

- y_s : Binary decision variable which is 1 if splitter s is selected and 0 otherwise.
- x_e : Binary decision variable which is 1 if edge e is selected and 0 otherwise.

2.3 Objective

Minimize the total cost, which includes the cost of selected splitters, the cost of laying fiber on selected edges, and the trenching cost.

3 Data

Listing 1: Python Code for PON Model

```
model pon

end

import elytica
import json
file = open('gars6.json')
graph = json.load(file)

edges = [{**item, 'nid': index} for index, item in
    enumerate(graph["edges"])]
nodes = [{**item, 'nid': index} for index, item in
    enumerate(graph["nodes"])]
loadSplitters = lambda: [n['nid']  for n in nodes if
    n['type'] == 'triangle']
loadONUs = lambda: [n['nid']  for n in nodes if n['type']
    == 'circle']
loadIntermediate = lambda: [n['nid']  for n in nodes if
    n['type'] == None]
loadCOs = lambda: [n['nid']  for n in nodes if n['type'] ==
    'square']
loadCommodities = lambda: loadONUs() + loadSplitters()
loadVertices = lambda: [n['nid'] for n in nodes]
loadEdges = lambda: [a['nid'] for a in edges]
loadM = len(edges) * 2
def generateArcs(edges):
  edge_len = len(edges)
  return [
    {
      'weight': edge['weight'],
      'source': direction[0],
      'target': direction[1],
      'id': edge['id'] + suffix,
```

```
      'label': edge['label'],
      'nid': edge['nid'] if index == 0 else edge['nid'] +
          edge_len
    }
    for edge in edges
    for index, (direction, suffix) in
        enumerate([((edge['source'], edge['target']), '-1'),
        ((edge['target'], edge['source']), '-2')])
  ]

arcs = generateArcs(edges)
loadArcs = lambda: [a['nid'] for a in arcs]
loadIncomming = lambda: {n['nid']: [a['nid'] for a in arcs
    if a['target'] == n['id']] for n in nodes}
loadOutgoing = lambda: {n['nid']: [a['nid'] for a in arcs
    if a['source'] == n['id']] for n in nodes}
loadArcWeights = lambda: {a['nid']:a['weight'] for a in
    arcs}
loadNodeCosts = lambda: {n['nid']: 0 if not n['modules']
    else n['modules'][0]['cost'] for n in nodes}


# Calculate the midpoint
mid_x = sum(node["x"] for node in nodes) / len(nodes)
mid_y = sum(node["y"] for node in nodes) / len(nodes)

# Translate the nodes relative to the midpoint
translated = [{"x": node["x"] - mid_x, "y": node["y"] -
    mid_y} for node in nodes]

# Find the scaling range
min_x, max_x = min(node["x"] for node in translated),
    max(node["x"] for node in translated)
min_y, max_y = min(node["y"] for node in translated),
    max(node["y"] for node in translated)

def get_color(node):
  if node["type"] == 'triangle':
    return '#0000ff'
  if node["type"] == 'square':
    return '#00ff00'
  if node["type"] == 'circle':
    return '#ff0000'
  return '#000000'

def main():
  solve = False # False will render the input

  visual_vertices = [{
    **node,
```

```
    "x": ((node["x"] - mid_x - min_x) / (max_x - min_x or 1)) * 500,
    "y": ((node["y"] - mid_y - min_y) / (max_y - min_y or 1)) * 500,
    "name": node["id"],
    "symbol": node["type"],
    "symbolSize": node["size"],
    "itemStyle": {"color": node["color"] if node["color"] is not None else get_color(node)},
    "id": None
   } for node in nodes]
  visual_edges = [{
    **edge,
    "id": None,
    "weight": None,
    "label": None,
    "symbol": None
   } for edge in edges]
  visual_vertices = [{k: v for k, v in node.items() if v is not None} for node in visual_vertices]
  visual_edges = [{k: v for k, v in edge.items() if v is not None} for edge in visual_edges]
  if solve:
    elytica.init_model("pon")
    elytica.run_model("pon")
    # visualize the resulting datasets with the following:
    #resulting_edges = [ e for e in elytica.get_model_set("pon", "E")  if elytica.get_variable_value("pon", f"x{e}") > 0.5 ]
    #resulting_splitters = [ s for s in elytica.get_model_set("pon", "S")  if elytica.get_variable_value("pon", f"y{s}") > 0.5 ]
    #no_splitters = [v for v in visual_vertices if ('type' not in v) or (v['type'] != 'triangle')]
    #splitters = [v for v in visual_vertices if v['nid'] in resulting_splitters]
    #visual_vertices  = splitters + no_splitters
    #visual_edges = [e for e in visual_edges if e['nid'] in resulting_edges]

  chart = {
    "series": [ {
      "roam": 'zoom',
      "type": 'graph',
      "layout": 'none',
      "data": visual_vertices,
      "edges": visual_edges,
      "lineStyle": {
        "opacity": 1,
        "width": 3
```

```
        }
      }
    ],
  }


  elytica.write_results(json.dumps({"chart": chart}))
  return 0
```

Participants will be provided with a starting code (as shown above) which includes a JSON data structure containing the network graph.

4 Evaluation

Solutions will be evaluated based on the following criteria:

- Correctness: Does the solution respect all given constraints?
- Optimality: How close is the solution to the optimal cost?
- Computational Efficiency: How efficiently does the solution run?

5 Submission Guidelines

Participants are encouraged to use Mixed-Integer Linear Programming (MILP) via Elytica to address the problem. However, if heuristics are employed, the implementation should be compilable on a Unix-based machine. For those using proprietary MIP solvers, a .lp file must be provided, which will be executed using HiGHS. All relevant code and results should be included in the submission.

6 Datasets

The intricacies of a JSON-formatted dataset are outlined in Listing 2. The dataset consists of two primary entities: `edges` and `nodes`.

Each `edge` represents a potential location for placing a fiber and is characterized by several attributes. The `weight` of an edge indicates its length. The unit cost of trenching is R150 per meter, while the cost of fiber is R50 per meter. This is used to calculate the total cost for each edge based on its length. Additionally, each edge has details about its starting point (`source`), endpoint (`target`), and a unique identifier (`id`).

The `nodes` encapsulate distinct network components with specified functionalities:

- CO (Central Office) nodes are symbolized as squares, acting as the primary hubs for telecommunications.

- Triangles pinpoint potential locations for splitters which, when positioned, would be responsible for distributing optical signals.
- ONUs (Optical Network Units) are visualized as circles and act as service subscriber endpoints in the network.
- Intermediate nodes, which might serve as routing points or simple connection hubs, don't have a distinct type but remain essential for maintaining the network's connectivity.

Listing 2: JSON Structure

```
{
    "edges": [
        {
            "weight": ...,
            "source": ...,
            "id": ...,
            "label": ...,
            "target": ...
        },
        ...
    ],
    "nodes": [
        {
            "color": ...,
            "size": ...,
            "modules": [
                {
                    "cost": ...,
                    "capacity": ...
                },
                ...
            ],
            "id": ...,
            "x": ...,
            "type": ...,
            "label": ...,
            "y": ...
        },
        ...
    ]
}
```

Figure 1 offers a visual representation of the `micronet.json` dataset, showcasing the interconnectedness of the nodes. The specific shape of each node allows for immediate identification of its role within the network, while the edges, underscore potential pathways for fiber and their corresponding lengths.

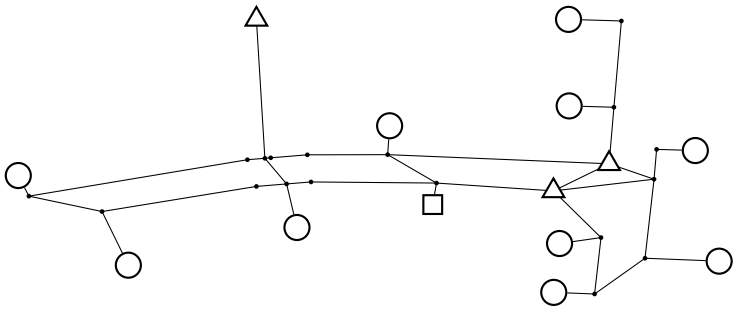

Figure 1: micronet.json visualized.

Competition Rules

1. **No Team Participation**: This is an individual competition. Team submissions will not be accepted.

2. **Time Limit**: Participants are given a strict time limit of 2 hours from the start of the competition to finalize and submit their solutions.

3. **Assistance**: The Elytica team will be available to assist with the Elytica syntax and general programming errors. However, they will not provide any assistance with model formulation.

4. **Submission**: Participants should email their script or code, along with the model formulation, to `info@elytica.com` before the deadline.

5. **Resources**: Participants are welcomed to make use of any online resources to aid their work during the competition. However, all work submitted must be original and not copied from other sources.