华为算法精英实战营第十五期

作品思路

获奖赛队名称：mine_qihang
所属赛题：华为算法精英实战营-第十五期：Rerouting in an optical network

1 赛题核心点介绍

1.1 赛题内容

一个光网络可以表示为图的形式。图中的每条边是光缆（光纤）。信息通过特定波长（频率）的光进行传输。光纤可以同时传输多个不同波长的信号。随着时间的推移，网络中的某些元件（光纤）可能会发生故障，我们需要为中断的服务（连接）找到新的路径。此外，网络中可能会发生随机的故障序列。任务的目标是**尽可能提供更多服务的正常运行**。核心挑战包括：

- **动态性**：需实时响应边删除请求，快速找到新路径和波长，避免与已有服务冲突。
- **资源竞争**：同一边不同服务必须使用不同波长，且初始路径的波长需永久保留。
- **长期影响**：当前路径选择需考虑后续请求的灵活性，避免过度阻塞未来可选路径。

1.2 取胜需要特别关注

- **对未来断边的考虑**。如果不考虑未来，只关注当前寻找路径的最优性，很可能会导致未来随机断边时影响的业务更多、更不易分配。
- **求解速度**。如果采用随机化算法，那么搜索的速度就要足够快，这样才能在有限的时间内搜索出更优的解。

2 整体思路介绍

从 baseline 到最终的代码主要有以下过程：

首先是实现分频率的 BFS 寻路算法。频率从小到大依次遍历，最先能找到路径的就直接返回。后面又修改成了把所有频率遍历一遍，然后选最短的返回。

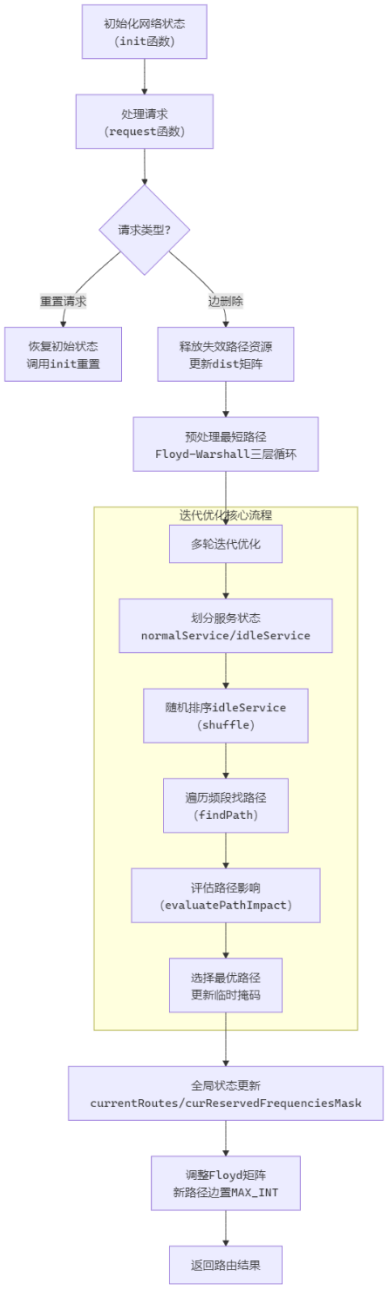
但显然上面的方法不够好，因为并不是路越短越好（当然这是一个重要的因素），所以我们还需要去考虑怎样的路径才是好的？我认为能让未来断边概率影响最小的路径是好的，所以我设计了一个影响评估函数 evaluatePathImpact()来对每条路径打分。打分相同时，路径越短越好。

这样看起来还不错了，但是我们在分配的过程中是按照断边内业务顺序划分

的，这会导致后续的业务明明可以分配成功的，但因为前面的业务占用了一些频率，使他们的关键路径被堵住。这可以通过同时对所有业务进行分配，但这样实现太复杂了，而且想找到在这个条件下的最优解是很难的。所以我使用了随机化算法，随机打乱业务顺序来多算几次，用其中最少失败业务数量的解来作为最后输出的结果（当最少业务数量相同时，考虑打分成绩和最大的）。

随机化算法最需要的就是求解速度，因为算的越快能搜索的解越多，所以我从题目给出的节点数、边数限制来考虑，可以用 Floyd 算法做一个预处理，从而规避掉一些根本算不出路径的 s-t 对。

综上，通过从 baseline 一步一步优化过来，最终取得了一个不错的结果。以下是代码流程。



3 核心策略详解

下面详解两个主要优化点。

3.1 路径评估函数

这个函数是整个算法中最核心的部分，会大幅影响分数。核心思想是：避免过度占用公共资源，以提升后续请求的成功率。具体来说，考虑了计算当前服务与其余空闲服务的路径重叠边数量。如下面代码所示：

```
for (int idx = 0; idx < consideredIdleService; ++idx) {
    int otherServiceId = idleService[idx] + 1;
    if (otherServiceId == serviceId) continue;
    // 检查该服务在该频段下是否可达
    if (currentDist[freq][s][t] == MAX_INT) continue;
    vector<int> otherPath = findPath(...);
    if (!otherPath.empty()) {
        int overlap = 0;
        for (int edgeId : otherPath) {
            if (blockedEdges.count(edgeId)) overlap++;
        }
        score1 -= 1.0 * overlap;
    }
}
```

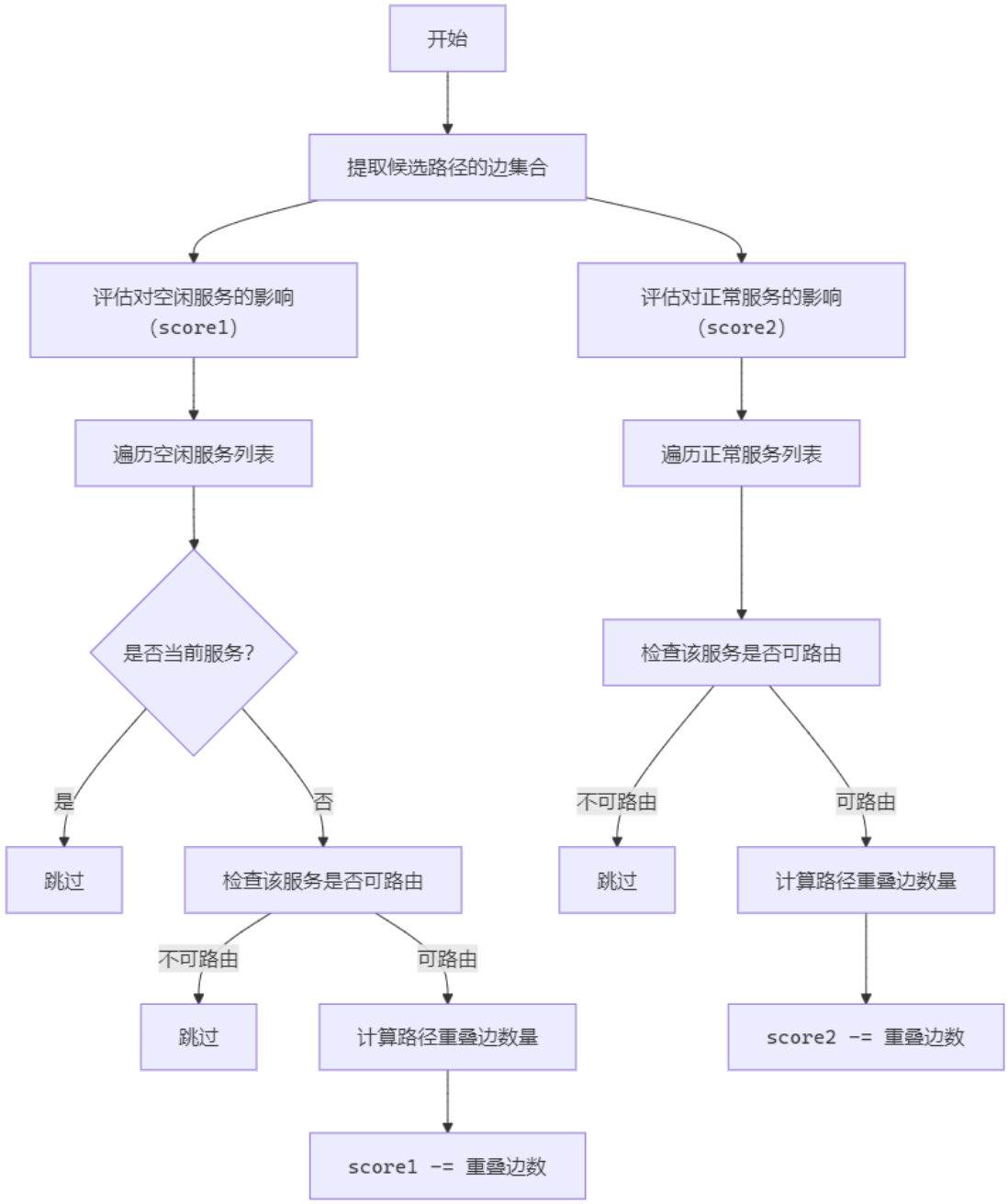
这是在预测候选路径对未恢复服务的潜在阻塞。同时仅遍历部分空闲服务（MAX_CONSIDERED_SERVICES），平衡效率与准确性。

然后也对正常服务影响进行评估：

```
for (int i = 0; i < consideredServices; ++i) {
    int otherServiceId = normalService[i] + 1;
    // 类似空闲服务处理，计算重叠边
    // ...
    score2 -= 1.0 * overlap;
}
```

主要目的是防止候选路径阻塞已恢复服务的备用路径（如未来需再次调整时）。

下面的示意图展示了函数的过程。



3.2 预处理

我使用了 Floyd-Warshall 算法做预处理，因为如果知道每个 s-t 对每个频率下是否能到达是很重要的，可以帮我们减少空耗。具体来说，通过四重循环计算各频段最短路径（currentDist 更新）：

```
for(int k=1; k<=N; k++)
   for(int i=1; i<=N; i++)
      for(int j=1; j<=N; j++)
         for(int freq=1; freq<=W; freq++)
```

```
            currentDist[freq][i][j] = min(...)
```

这个时间复杂度为$O(WN^3)$，这在题目要求条件下是可以满足时限要求的。

然后在找完路径返回前更新：

```
for(int i = 0; i < K; ++i)
  if (currentRoutes[i].p.empty() && !bestRoutes[i].p.empty())
    for(int edgeId : bestRoutes[i].p)
      Edge &edge = edges[edgeId - 1];
      dist[bestRoutes[i].w][edge.u][edge.v] = MAX_INT;
      dist[bestRoutes[i].w][edge.v][edge.u] = MAX_INT;
```

综上，我的解题思路按照从问题中挖掘潜在特性和逐步优化简单算法的方式展开。