Appendix

A. *MulGraphPulse algorithm*

The MulGraphPulse algorithm is to apply the Pulse algorithm on the layer graph.
If we have already generated the layer graph $\overline{G}$, we can almost disregard the sequence $\mathcal{T}$ because this constraint is implied by the properties of the layer graph. First, we discusses how to make sure the path tour of $\mathcal{T}$ is elementary and how the method influences the Pruning by Dominance strategy. In Section III-C, we discusses that the optimal path we find does not have any loops. However, if we search in the new graph $\overline{G}$, it is not enough to guarantee that the resulting path is elementary. Actually node $v^0, v^1, ..., v^n$ in $\overline{G}$ represent the same node v in original graph G. Therefore, even though we can be sure that there are no loops in the optimal path $\overline{P}$ found using the Pulse algorithm, we cannot guarantee that the path tour we obtain from $\overline{P}$ is elementary. To address this issue, there are two methods available:

1) Visited node record: To record whether a node has been visited, we can use a Visited array. It is important to note that this Visited array requires special operations. Typically, we need to ensure that the nodes $v^0, v^1, \ldots, v^n$ are not simultaneously visited unless the nodes are on the edges that connect different layers. Because when we get the path tour from $\overline{P}$, the edges connecting different layers will be eliminated. Algorithm 1 outlines the process for checking visited nodes. We use a Visited array $\mathcal{V}$ to track this, the length of $\mathcal{V}$ is equal to the size of original graph node set. The value of $\mathcal{V}[v] \in \{0, 1, 2\}$, where $\mathcal{V}[v] = 0$ if v has not been visited, $\mathcal{V}[v] = 1$ if v has been visited by a node that belongs to the connecting edges, and $\mathcal{V}[v] = 2$ if v has been visited by a node that does not belong to the connecting edges.

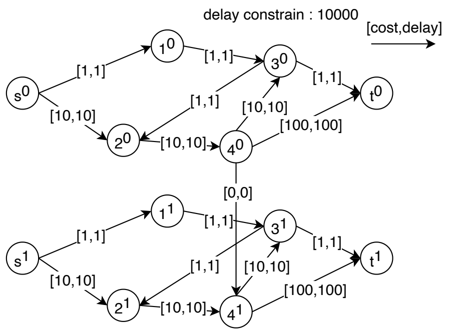

Fig. 1. Pruning by dominance becomes disabled in layer graph.

If we use this method, the Pruning by Dominance strategy is also disabled. Figure 1 illustrates this. Consider two path $P_1 = s^0 \to 1^0 \to 3^0 \to 2^0 \to 4^0 \to 4^1$ and $P_2 = s^0 \to 2^0 \to 4^0 \to 4^1$. If we use pruning by Dominance strategy, P_2 will be pruned. But P_2 can visit 3^1 next but P_1 cannot because P_1 has visited 3^0. And $P_2 \to 3^1 \to t^1$ is better than $P_1 \to t^1$. Obviously, Pruning by Dominance strategy make us lose a better result.

Algorithm 1: Visited check

```
Input: Original graph $G(V, E)$, layer graph $\overline{G}(\overline{V}, \overline{E})$,
       Visited array $\mathcal{V}$, node $v$
Output: the updated Visited array $\mathcal{V}$, is the node $v$
        can be visited
Use $F$ to denote whether the node $v$ can be visited,
  Initialize $F = True$.
$v_{\text{original}} \leftarrow$ the original node of $v$ in G.
if $\mathcal{V}[v_{original}]$ == 2 then
    $F \leftarrow False$;
end
else
    if $\mathcal{V}[v_{original}]$ == 1 then
        if $v$ is not the node on the connecting edges
          then
            $F \leftarrow False$;
        end
    end
    else
        if $v$ is the node on the connecting edges then
            $\mathcal{V}[v_{\text{original}}] \leftarrow 1$;
        end
        else
            $\mathcal{V}[v_{\text{original}}] \leftarrow 2$;
        end
    end
end
return $\mathcal{V}$, $F$;
```

2) divide and conquer: An alternative approach is to not track the visited nodes during the search and instead check for elementary paths only after the search has finished. The main concept is to avoid additional operations during the search to ensure that the final path tour is elementary. Instead, we check for this property only after the search has finished. If the path is elementary, then we obtain the final result. If not, we need to disable the repeated nodes respectively, and search again until we find an elementary path, and we choose the best result of all new graphs as optimal result. However, this method may result in a large number of sub-problems, which can decrease the algorithm's efficiency. So we don't choose this strategy.

B. *Extension to another variant: adding Forward constraint*

The MPulse algorithm and MulGraphPulse algorithm are proposed in Section IV and Appendix A, respectively, to solve the DCSPT problem. In this section, they will be modified to adapt to DCFSPT problem, i.e. adding Forward constraint into DCSPT.

1) MPulse algorithm: DCFSPT problem requires that a path P can only pass through the nodes in subset T_i if all preceding subsets $T_1, ..., T_{i-1}$ have been visited. Therefore, in the MPulse+ algorithm, not only do we update the mask, but we also need to prune the sub-paths that visit nodes out of order with respect to the forward constraint. This requires designing pruning strategies to prevent the depth-first search process from searching paths that do not follow the required order.

According to the Forward constraint requirement, a sub path P needs to satisfy the following conditions when visiting nodes: given a sequence $\mathcal{T} = \{T_1, T_2, ..., T_n\}$, a sub path P can only visit node v if $P.\mathcal{M} + 1 \geq v.weight$. The reason is as follows: A sub path P has visited a subset of node $I = \{v_1^I, v_2^I, ..., v_i^I\}$ such that $v_j^I \in T_j$ for $j \in \{1, 2, \ldots, i\}$ in order. If a node v that satisfies $P.\mathcal{M} + 1 < v.weight$, then v must belong to T_k for some $k \in \{i+2, ..., n\}$. And according to the requirement of DCFSPT problem, the sub path can not search v, thereby violating the Forward constraint.

Then the Pruning by Mask is as follows:

```
if $P.\mathcal{M} + 1 < v$.weight
    $continue$;                                   (1)
end
```

By adding the Pruning by Mask before line 21 of Algorithm 3, we generate the MPulse+ algorithm, which can be used to solve the DCFSPT problem.

Algorithm 2: Adjust graph

```
Input: layer graph $\overline{G}(\overline{V}, \overline{E})$, sequence
       $\mathcal{T} = \{T_1, T_2, ..., T_n\}$
Output: the adjusted graph
1 for every i from $\{2, 3, ..., n\}$ do
2     for every node $v$ in $T_i$ do
3         delete $\{v^0, ..., v^{i-2}\}$ from $\overline{G}$;
4     end
5 end
6 return $\overline{G}$;
```

2) MulGraphPulse algorithm: We can explain the requirements of DCFSPT problem on the layer graph as follows: the layer that a sub path has visited is related to the subset the sub path has visited. For example, if a sub path searches the ith layer, then it has visited some nodes $v_1^I, v_2^I, dots, v_{i-1}^I$ such that $v_j^I \in T_j$, $j \in \{1, ..., i-1\}$. Therefore, for a sequnce $\mathcal{T} = \{T_1, T_2, ..., T_n\}$, the nodes in subset T_j can only be visited after all preceding subsets $T_1, ..., T_{i-1}$ have been visited, that means node $v \in T_i$ cannot be visited in $\{0, ..., i-2\}$ layers, where $i \in \{2, ..., n\}$. If we want to modify MulGraphPulse algorithm to adapt to DCFSPT problem, we only need to delete specific nodes in specific layers. This is shown in Algorithm 2.

Once we obtain the adjusted graph, we can apply the solutions discussed in Appendix A to solve the DCFSPT problem.

C. Extension to another variant: relaxing elementary constraint

Modifying the algorithm for this variant is a straightforward process that involves removing the "visit node record" operation from either MPulse or MulgraphPulse.