A Modified SEN-PANDA

Consider the SEN-PANDA algorithm in Algorithm 2 in the main text. If certain applications allow overlapping matching subgraphs for different query components, then this algorithm can be modified as follows to handle such scenario. The modified algorithm is shown in Algorithm 1.

- Remove Lines 1 and 5-14.
- Lines 15-18 in the original algorithm are modified to Lines 4-7 (Algorithm 1).

Algorithm 1: The modified SEN-PANDA algorithm

```
Input: A network $G = (V, E)$, a partial topology query
       $Q_P = (q_1, \ldots, q_\ell)$, a positive integer $k$.
Output: Top-$k$ partial topology-based matches of $Q_P$.
1 for $i \in \{1, \ldots, \ell\}$ do
2     $\text{SM}_i \leftarrow \text{ISOMORPHIC}(q_i, G)$
3 $H \leftarrow \text{GENERATEMERGEGRAPH}(G, \{\text{SM}_1, \ldots, \text{SM}_\ell\})$
4 for $i \in \{1, \ldots, \ell\}$ do
5     $V_i \leftarrow \{s | s \in H_s \wedge i \in \Upsilon_s\}$
6 $R \leftarrow \text{GST}(H, \{V_1, \ldots, V_\ell\}, k)$
7 return $R$
```

B Modified PO-PANDA

The original PO-PANDA algorithm (Algorithm 3) in the main text can be modified in the following ways to handle the case of overlapping matching subgraphs for different query components. The modified algorithms are shown in Algorithms 2 and 3.

- Remove Lines 12-13 because *sharedlabel* is not needed.
- Remove the check condition block (no need to check as all merged nodes are safe) of Line 20.
- Remove Lines 25-28 because there is no extra cost in the inner graph of any merged node.
- In Algorithm 4 in the main text, remove Lines 12-13 and Lines 15-21 as *sharedlabel* is not needed.

Algorithm 2: The modified PO-PANDA Algorithm

```
Input: A network $G = (V, E)$, a partial topology query
       $Q_P = (q_1, \ldots, q_\ell)$, a positive integer $k$.
Output: Top-$k$ partial topology-based matches of $Q_P$.
1  Initialize a heap $TreesHeap \leftarrow \emptyset$;
2  for $i \in \{1, \ldots, \ell\}$ do
3      $\text{SM}_i \leftarrow \text{ISOMORPHIC}(q_i, G)$;
4  $H \leftarrow \text{GENERATEMERGEGRAPH}(G, \{\text{SM}_1, \ldots, \text{SM}_\ell\})$;
5  $\mathbb{S} \leftarrow \text{GETMERGENODES}(H)$;
6  Initialize $\ell$ LMQs $\{\mathcal{P}_1, \ldots, \mathcal{P}_\ell\}, \mathcal{P}_i \leftarrow \emptyset$;
7  for $s \in \mathbb{S}$ do
8      for $i \in \Upsilon_s$ do
9          $\text{ENQUEUE}(s, 0, \mathcal{P}_i)$;
10         $s.parent(i) \leftarrow s$;
11         $s.source(i) \leftarrow s$;
12 while $\exists \mathcal{P}_i \neq \emptyset$ do
13     for $i \in \ell, \mathcal{P}_i \neq \emptyset$ do
14         $(v, d_s) \leftarrow \text{DEQUEUE}(\mathcal{P}_i)$;
15         if $i \notin \Upsilon_v$ then
16             $\text{ADD}(i, \Upsilon_v)$;
17         if $|\Upsilon_v| = \ell$ then
18             Generate a result tree $T$ using $v$;
19             Compute $cost(T)$;
20             $\text{ADD}(T, cost(T), TreesHeap)$;
21             if $\text{SIZE}(TreeHeap) \geq k$ then
22                 $R \leftarrow \text{GETTOPK}(TreeHeap)$;
23                 Replace merged nodes in $R$ with the
                   corresponding inner graphs;
24                 return $R$
25         $\text{NEIGHBOREXPLORATION}(v, i)$ /* Algorithm 3 */;
```

Algorithm 3: Modified NEIGHBOREXPLORATION procedure.

```
Input: A current node $v \in H$, an index $i$ of LMQ.
1  for $(u, (v, u).weight) \in v.edgesList()$ and $u \neq v.parent(i)$
   do
2      if $i \notin \Upsilon_u$ then
3          if $u \in \mathcal{P}_i$ then
4              if $(v, u).weight + dist(v) < dist(u)$ then
5                  $u.source(i) \leftarrow v.source(i)$;
6                  $u.parent(i) \leftarrow v$;
7                  $\text{UPDATE}(\mathcal{P}_i, u, (v, u).weight + dist(v))$;
8                  update $u.sharedlabels$;
9          else
10             $u.source(i) \leftarrow v.source(i)$;
11             $u.parent(i) \leftarrow v$;
12             $\text{ENQUEUE}(\mathcal{P}_i, u, (v, u).weight + dist(v))$;
```