A Companion Technical Report of "Towards Efficient Large-Scale Network Slicing: An LP Dynamic Rounding-and-Refinement Approach"

Wei-Kun Chen, Ya-Feng Liu, Fan Liu, Yu-Hong Dai, and Zhi-Quan Luo

In this report, we first give the proof of Theorem 1 in [1] and prove the hardness of problem (24) in [1] in Sections I and II, respectively. Then, we present the details of the inflation optimization (IO), the particle swarm optimization (PSO), and LP one-shot rounding (LPoR) algorithms, which are adapted from [2], [3], and [4], respectively, in Section III. Subsequently, we present the details of an LP static rounding algorithm for solving the VNF placement subproblem in Section IV. Finally, we provide more simulation results on the performance of the proposed LP relaxation (LP-II) and the proposed LP dynamic rounding-and-refinement (LPdRR) algorithm in [1] in Section V.

I. Proof of Theorem 1 in [1]

In this section, we give the proof of Theorem 1 in [1]. We first state the theorem as follows.

Theorem 1. *(i) Problem (MILP) is NP-hard even when there is only a single service and there is only a single function in the SFC of this service. (ii) Problem (MILP) is strongly NP-hard even when each node's capacity, link's capacity, and service's E2E delay threshold are infinite.*

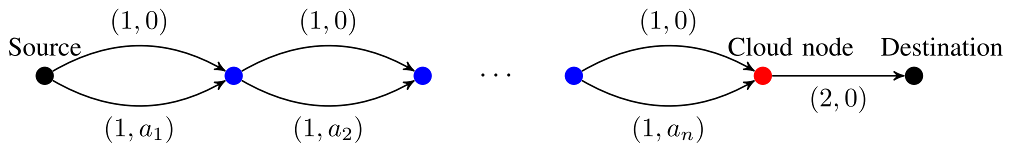


Fig. 1. The constructed network to prove Theorem 1(i), where the pair (c,d) over each link denotes that the communication capacity and the communication delay of the corresponding link are c and d, respectively.

Proof of Theorem 1(i)

We prove that checking the feasibility of problem (MILP) with a single service and a single function in the service's SFC is as hard as the partition problem, which is NP-complete [5]. Next, we introduce the partition problem: given a finite set $\mathcal{N} = \{1, \ldots, n\}$ of n elements and a size $a_i \in \mathbb{Z}_+$ for the i-th element with $\sum_{i \in \mathcal{N}} a_i = 2b$, does there exist a partition $\mathcal{N} = \mathcal{N}_1 \cup \mathcal{N}_2$ such that $\sum_{i \in \mathcal{N}_1} a_i = \sum_{i \in \mathcal{N}_2} a_i = b$?

Given any instance of the partition problem, we construct an instance of problem (MILP) with a single service and a single function in the service's SFC as follows.

- The set of network nodes $\mathcal{I}$ is $\{0, n+1\} \cup \mathcal{N}$ where node n is the cloud node that can process function f with the NFV delay 0. For each $i = 1, \ldots, n$, nodes $i-1$ and i are connected with two links with communication delays 0 and a_i, respectively, and the communication capacities of both links are all set to 1. To distinguish with the two links, we denote the one with communication delay 0 and the one with communication delay a_i as $\ell_{i-1,i}$ and $\ell'_{i-1,i}$, respectively. Moreover, nodes n and $n+1$ are connected by one link with the communication capacity and the communication delay being 2 and 0, respectively. See Fig. 1 for an illustration of the constructed network.
- There is only a single service that needs to be supported by the network. The service's SFC is $\mathcal{F} = \{f\}$, its data rate is $\lambda_0(1) = \lambda_1(1) = 2$, its E2E threshold is b, and its source and destination nodes are 0 and $n+1$, respectively.

In the following, we prove that the above constructed instance of problem (MILP) is feasible if and only if the answer to the partition problem is yes.

We first prove that if the partition problem has a "yes" answer, then the constructed instance of problem (MILP) is feasible. Suppose that there exist two subsets $\mathcal{N}_1$ and $\mathcal{N}_2$ such that $\mathcal{N}_1 \cup \mathcal{N}_2 = \mathcal{N}$ and $\sum_{i \in \mathcal{N}_1} a_i = \sum_{i \in \mathcal{N}_2} a_i = b$. We construct two paths p_1 and p_2 from nodes 0 to $n+1$ with unit data rate using the following links:

- p_1: $\ell_{i-1,i}$ for $i \in \mathcal{N}_1$, $\ell'_{i-1,i}$ for $i \in \mathcal{N}_2$, and $(n, n+1)$;
- p_2: $\ell'_{i-1,i}$ for $i \in \mathcal{N}_1$, $\ell_{i-1,i}$ for $i \in \mathcal{N}_2$, and $(n, n+1)$.

Apparently, such a traffic flow has an E2E delay b, and thus satisfies the E2E requirement of the service. As a result, the constructed instance of problem (MILP) is feasible.

To prove the other direction, suppose that problem (MILP) under the above construction has a feasible solution. By feasibility, the associated communication delay cannot be larger than b. In addition, for all $i \in \mathcal{N}$, the data rate on links $\ell_{i-1,i}$ and $\ell'_{i-1,i}$ are all 1 since the communication capacity of these links are all 1 and the data rate sent from node 0 to node $n+1$ must be equal to 2. This implies that the associated traffic flow of this solution can be partitioned into two paths p_1 and p_2. Let $\mathcal{N}_1 := \{i \in$

$\mathcal{N} : \ell'_{i-1,i}$ is on path $p_1\}$. Then, for all $i \in \mathcal{N}_2 := \mathcal{N} \backslash \mathcal{N}_1$, link $\ell'_{i-1,i}$ must be on path p_2. Since the communication delay of paths p_1 and p_2 cannot be larger than b, we have

$$\sum_{i \in \mathcal{N}_1} a_i \leq b \text{ and } \sum_{i \in \mathcal{N}_2} a_i \leq b. \tag{1}$$

This, together with the fact that $\sum_{i \in \mathcal{N}} a_i = 2b$, implies that

$$\sum_{i \in \mathcal{N}_1} a_i = \sum_{i \in \mathcal{N}_2} a_i = b. \tag{2}$$

Therefore, the answer to the partition problem is yes.

Finally, the above transformation can be done in polynomial time. Since the partition problem is NP-complete, we conclude that problem (MILP) with even a single service and a single function in the service's SFC is NP-hard. $\square$

Proof of Theorem 1(ii)

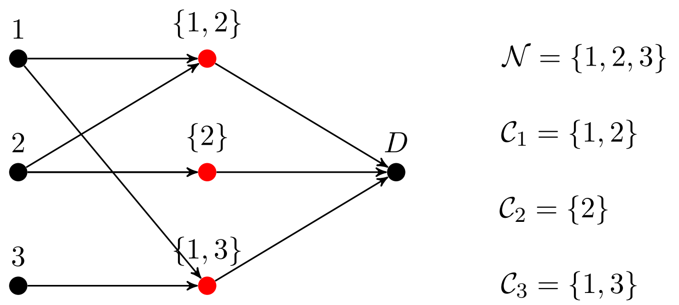
We prove the strong NP-hardness of problem (MILP) with infinite node and link capacities and infinite E2E delay threshold by establishing a polynomial time reduction from the minimum set covering problem, which is strongly NP-hard [5]. Next, we introduce the decision verison of the minimum set covering (D-MSC) problem: let $\mathcal{N}$ be a finite set of n elements and given m subsets $\mathcal{C}_j \subseteq \mathcal{N}$, $j = 1, \ldots, m$, does there exist $\mathcal{M}' \subseteq \mathcal{M} := \{1, \ldots, m\}$ such that $|\mathcal{M}'| \leq m'$ and $\cup_{j \in \mathcal{M}'} \mathcal{C}_j = \mathcal{N}$?

Given any instance of the D-MSC problem, we construct an instance of problem (MILP) with the conditions listed in the statement of Theorem 1(ii) as follows.

- The set of network nodes $\mathcal{I}$ is $\mathcal{N} \cup \mathcal{M} \cup \{D\}$, where $\mathcal{N}$ is the set of nodes that are the source nodes of corresponding services, $\mathcal{M}$ is the set of cloud nodes which can process function f, and D is the destination node of all services. The set of links $\mathcal{L}$ consists of two parts: (i) for each $i \in \mathcal{N}$ and $j \in \mathcal{M}$, $(i, j) \in \mathcal{L}$ if $i \in \mathcal{C}_j$; and (ii) for each $j \in \mathcal{M}$, $(j, D) \in \mathcal{L}$. See Fig. 2 for an illustration of this construction.
- For each $k \in \mathcal{N}$, we define a service k with the source and destination pair (k, D) and the SFC $\mathcal{F}(k) = \{f\}$. Hence, there are in total $|\mathcal{N}|$ services.

In the following, we prove that the above constructed instance of problem (MILP) has a feasible solution with at most m' activated cloud nodes if and only if the answer to the D-MSC problem is yes.

We first prove that if the D-MSC problem has a "yes" answer, then the constructed instance of problem (MILP) has a feasible solution with at most m' activated cloud nodes. Suppose that there exists $\mathcal{M}' \subseteq \mathcal{M}$ such that $|\mathcal{M}'| \leq m'$ and $\cup_{j \in \mathcal{M}'} \mathcal{C}_j = \mathcal{N}$. Let cloud node j, $j \in \mathcal{M}'$, be activated. Then, for each service k, we can place its SFC's function at a cloud node, namely j, where there exists a link between node



1
2
3
$\{1,2\}$
$\{2\}$
$\{1,3\}$
D
$\mathcal{N} = \{1,2,3\}$
$\mathcal{C}_1 = \{1,2\}$
$\mathcal{C}_2 = \{2\}$
$\mathcal{C}_3 = \{1,3\}$

Fig. 2. A toy example for illustrating the transformation of the D-MSC problem into the constructed network.

k and node j, i.e., $k \in \mathcal{C}_j$. Notice that such placement exists for all services since $\cup_{j \in \mathcal{M}'} \mathcal{C}_j = \mathcal{N}$. Therefore, we can place the functions in the SFCs of the $|\mathcal{N}|$ flows at these $|\mathcal{M}'|$ cloud nodes. Notice that, for each service, we can always find a traffic routing strategy since the communication capacity is infinite for each link. As a result, we obtain a feasible solution with at most m' activated cloud nodes.

Now we prove that if the constructed instance of problem (MILP) has a feasible solution with at most m' activated cloud nodes, then the D-MSC problem has a "yes" answer. Suppose that problem (MILP) under the above construction has a feasible solution with at most m' activated cloud nodes. Let $\mathcal{M}'$ be the set of activated cloud nodes (and $|\mathcal{M}'| \leq m'$). By construction, each flow $k \in \mathcal{N}$ must be placed at one of the cloud nodes in $\mathcal{M}'$, namely, cloud node j. Moreover, to guarantee the existence of a traffic routing strategy, there must exist a link between node k and node j, i.e., $k \in \mathcal{C}_j$. Combining the above implies that $\cup_{j \in \mathcal{M}'} \mathcal{C}_j = \mathcal{N}$. Therefore, the answer to the D-MSC problem is yes.

Finally, the above transformation can be done in polynomial time. Since the D-MSC problem is strongly NP-complete, we conclude that problem (MILP) is strongly NP-hard when all nodes' capacity, links' capacity, and service's E2E delay threshold are infinite. □

II. Hardness of problem (24) in [1]

Proposition 1. *Checking the feasibility of problem (24) with $\lambda_1(k) \geq \frac{1}{3}\mu_v$ for all $k \in \mathcal{K}$ and $v \in \mathcal{V}(k)$ is strongly NP-complete.*

Proof. We prove that checking the feasibility of problem (24) with $\lambda_1(k) \geq \frac{1}{3}\mu_v$ for all $k \in \mathcal{K}$ and $v \in \mathcal{V}(k)$ is as hard as the 3-partition problem, which is NP-complete [5]. We first introduce the 3-partition problem: given a finite set $\mathcal{N} = \{1, \ldots, 3n\}$ of $3n$ elements and a size $a_k \in \mathbb{Z}_+$ for the k-th element with $\frac{B}{4} < a_k < \frac{B}{2}$ and $\sum_{k=1}^{3n} a_k = nB$, can $\mathcal{N}$ be partitioned into n disjoint sets $\mathcal{N}_1, \ldots \mathcal{N}_n$ such that $\sum_{k \in \mathcal{N}_v} a_k = B$, $1 \leq v \leq n$?

Given any instance of the 3-partition problem, we construct an instance of problem (24) with $\lambda_1(k) \geq \frac{1}{4}\mu_v$ for all $k \in \mathcal{K}$ and $v \in \mathcal{V}(k)$ by setting $\mathcal{K} = \mathcal{N}$, $\mathcal{V} = \{1, \ldots, n\}$, $\lambda_1(k) = a_k$ and $\mathcal{V}(k) = \mathcal{V}$ for all $k \in \mathcal{K}$, and $\mu_v = B$ for all $v \in \mathcal{V}$. In the following, we will show that the answer to the given instance of the 3-partition problem is true if and only if problem (24) with the above constructed parameters has a feasible solution.

Suppose that $\mathcal{N}$ can be partitioned into n subsets $\mathcal{N}_1, \ldots, \mathcal{N}_n$ such that $\sum_{k \in \mathcal{N}_v} a_k = B$, $1 \leq v \leq n$. We construct a point $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ by setting $\bar{y}_v = 1$ for all $v \in \mathcal{V}$, $\bar{x}_{v,1}(k) = 1$ for $k \in \mathcal{N}_v$ and $v \in \mathcal{V}$ and $\bar{x}_{v,1}(k) = 0$ for the others. Clearly, $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is a feasible solution of problem (24).

Now suppose that problem (24) has a feasible solution $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$. Summing up Eq. (3) for $v \in \mathcal{V}$ and using Eq. (1) in [1], we obtain

$$\sum_{k \in \mathcal{K}} \lambda_1(k) = \sum_{k=1}^{3n} a_k \leq B \sum_{v=1}^{n} y_v \leq nB.$$

This, together with $\sum_{k=1}^{3n} a_k = nB$, implies that $y_v = 1$ for all $v \in \mathcal{V}$ and (3) in [1] holds with equality at point $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ for all $v \in \mathcal{V}$. As a result, we must have

$$\sum_{k=1}^{3n} a_k \bar{x}_{v,1}(k) = B, \ \forall\ v \in \mathcal{V}.$$

Let $\mathcal{N}_v = \{k \mid \bar{x}_v(k) = 1, \ 1 \leq k \leq 3n\}$. Then, it follows that

$$\sum_{k \in \mathcal{N}_v} a_k = \sum_{k=1}^{3n} a_k \bar{x}_{v,1}(k) = B, \ \forall\ v \in \mathcal{V}.$$

By (2) in [1], $\mathcal{N}_1, \ldots, \mathcal{N}_n$ are disjoint, implying that the answer to the given 3-partition instance is yes.

Finally, the above transformation can be done in polynomial time. Since the 3-partition problem is strongly NP-complete, we conclude that checking the feasibility of problem (24) with with $\lambda_1(k) \geq \frac{1}{3}\mu_v$ for all $k \in \mathcal{K}$ and $v \in \mathcal{V}(k)$ is also strongly NP-complete. □

III. Detailed Description of IO, PSO, and LPoR

In this section, we present the IO, PSO, and LPoR algorithms, which are adapted from [2], [3], and [4], respectively. The three algorithms first use different procedures to map VNFs into cloud nodes and then find paths connecting two adjacent VNFs in the network by solving a multicommodity flow problem, which can be solved in polynomial time [6, Chapter 17]. Next, we shall discuss the procedures of the three algorithms to map VNFs into cloud nodes in details.

IO maps VNFs into cloud nodes using an inflation optimization procedure, i.e., maps the services' VNFs into cloud nodes one by one. The details of IO are summarized in Algorithm 1 in this report.

Algorithm 1 The IO algorithm for solving the network slicing problem

```
1: Sort the services according to their total data rates, i.e., $\sum_{s\in\mathcal{F}(k)} \lambda_s(k)$, $k \in \mathcal{K}$;
2: for each service $k$ do
3:     Let $s \leftarrow 1$ be the index of the first function of service $k$ that has not been processed;
4:     while ($s \leq \ell_k$) do
5:         if (there exists some activated node that can process function $f_s^k$ and has a sufficient remaining computational capacity) then
6:             Select an activated node with the largest $t$ such that functions $f_s^k, \ldots, f_t^k$ can be processed at this node;
7:             Update $s \leftarrow t+1$ and the remaining node capacity;
8:         else if (there exists some inactivated node that can process function $f_s^k$ and has a sufficient remaining computational capacity) then
9:             Select an inactivated node with the largest $t$ such that functions $f_s^k, \ldots, f_t^k$ can be processed at this node;
10:            Mark this node as an activated node and update $s \leftarrow t+1$ and the remaining node capacity;
11:        else
12:            Stop and declare that the algorithm fails to find a feasible solution.
13:        end if
14:    end while
15: end for
16: If the above procedure finds a feasible solution for the VNF placement subproblem, solve the traffic routing subproblem by solving the multicommodity flow problem.
```

LPoR solves the LP relaxation of the network slicing problem to obtain its solution and then uses a one-shot rounding strategy to map VNFs into cloud nodes based on this solution. The details of LPoR are summarized in Algorithm 2 in this report.

In the PSO for the network slicing problem, a particle is represented as a potential solution for the VNF placement subproblem, flying through the solution space by following the current best solutions. Each particle relates to two vectors, called *position vector* and *velocity vector*. Position vector $\boldsymbol{X}$ denotes a VNF placement strategy, that is, $X_s^k = v$ if the s-th function of service k, f_s^k, is placed at cloud node v. Velocity vector $\boldsymbol{V} \in \{0,1\}^{|\mathcal{K}|}$ denotes the status for updating the VNF placement strategy $\boldsymbol{X}$: $V^k = 0$ if the VNF placement strategy X for service k is required to be updated; and $V^k = 1$ otherwise. At the

Algorithm 2 The LPoR algorithm for solving the network slicing problem

```
1: Solve problem (LP-II) to obtain its solution $(\boldsymbol{x}^*, \boldsymbol{y}^*, \boldsymbol{r}^*, \boldsymbol{\theta}^*)$ and sort $\{x^*_{v,s}(k)\}$ in the descending order;
2: for each $v \in \mathcal{V}$, $k \in \mathcal{K}$, and $s \in \mathcal{F}$ (according to the descending order of $\{x^*_{v,s}(k)\}$) do
3:   if function $f_s^k$ has not been mapped into a cloud node and cloud node $v$ has a sufficient remaining computational capacity then
4:     Map function $f_s^k$ into cloud node $v$;
5:     Mark function $f_s^k$ as a mapped function and update the remaining capacity of cloud node $v$;
6:   end if
7: end for
8: If the above procedure finds a feasible solution for the VNF placement subproblem, solve the traffic routing subproblem by solving the multicommodity flow problem.
```

beginning, PSO randomly initializes a set of particles, denoted by $\{1, \ldots, P\}$. Then, in each step of the evolutionary process, the velocity and position of particle p are updated as follows:

$$V^k(p) \leftarrow r_1 V^k \oplus r_2 \left(\boldsymbol{B}^k(p) \ominus \boldsymbol{X}^k(p) \right) \oplus r_3 \left(\boldsymbol{G}^k \ominus \boldsymbol{X}^k(p) \right), \tag{3}$$

$$\boldsymbol{X}^k(p) \leftarrow \boldsymbol{X}^k(p) \otimes V^k(p), \quad \forall\, k \in \mathcal{K}, \tag{4}$$

where

- $\boldsymbol{B}(p)$ denotes the position with the best fitness (i.e., the smallest number of activated cloud nodes) found so far for the p-th particle,
- $\boldsymbol{G}$ denotes the position with the best fitness in the swarm,
- $\boldsymbol{B}^k(p) \ominus \boldsymbol{X}^k(p)$ ($\boldsymbol{G}^k \ominus \boldsymbol{X}^k(p)$, respectively) is equal to 0 if $B_s^k(p) \neq X_s^k(p)$ ($G_s^k \neq X_s^k(p)$, respectively) for some $s \in \mathcal{F}(k)$, and is equal to 1 if $B_s^k(p) = X_s^k(p)$ ($G_s^k = X_s^k(p)$, respectively) for all $s \in \mathcal{F}(k)$,
- $r_1, r_2, r_3 \in [0, 1]$ satisfy $r_1 \leq r_2 \leq r_3$ and $r_1 + r_2 + r_3 = 1$,
- $r_1 a \oplus r_2 b \oplus r_3 c$ takes value a, b, or c with probabilities r_1, r_2, or r_3, respectively, and
- $\boldsymbol{X}^k(p) \otimes V^k(p)$ takes $\boldsymbol{X}^k(p)$ if $V^k(p) = 1$; otherwise, $\boldsymbol{X}^k(p)$ will be randomly updated to a new solution (i.e., the functions in service k will be randomly reallocated to the cloud nodes).

Following [3], we construct or update the position vectors of the particles taking into account the constraints and objective function of the network slicing problem. Specifically, we first sort the services (whose functions have not been allocated to the cloud nodes) according to their total data rates

$\sum_{s\in\mathcal{F}(k)}\lambda_s(k)$, $k\in\mathcal{K}$, and then sequentially place the functions of the services at the cloud nodes. In addition, when processing function f_s^k, we first compute the score of each cloud node v:

$$\text{Score}_v = S_v \text{A}_v t_v + (1-S_v)\text{IA}_v t_v, \tag{5}$$

where

- S_v denotes the current activation status of cloud node v, i.e., $S_v = 1$ if cloud node v is activated; otherwise, $S_v = 0$,
- t_v denotes the maximum number of functions (from f_s^k) that can be hosted at cloud node v, i.e., functions $f_s^k,\ldots,f_{s+t_v-1}^k$ can be simultaneously hosted at cloud node v but functions $f_s^k,\ldots f_{s+t_v}^k$ cannot be simultaneously hosted at cloud node v (due to, e.g., the limited capacity of cloud node v),
- A_v and IA_v are the weights for the cloud node v satisfying $\text{A}_v > \text{IA}_v > 0$.

Then, we place functions $f_s^k,\ldots,f_{s+t_{v_0}-1}^k$ at cloud node $v_0\in\mathcal{V}$ with probability $\frac{\text{Score}_{v_0}}{\sum_{v\in\mathcal{V}}\text{Score}_v}$. The above strategy tends to place multiple (adjacent) functions of a service into an activated cloud node. It enjoys the following two advantages: (i) it tends to choose a VNF placement strategy with a small number of activated cloud nodes and (ii) it implicitly takes the E2E delay of the services into consideration (as the E2E delay of a service tends to be small if multiple (adjacent) functions are processed at the same cloud node).

We summarize the procedure for placing the functions of a service and the overall procedure for PSO in Algorithms 3 and 4, respectively. In our experiments, we choose $r_1 = 0.1$, $r_2 = 0.2$, $r_3 = 0.7$, $\bar{r} = 0.1$ (the probability of setting $\boldsymbol{V}^k(p)$ to be 0), $\text{IterLim} = 100$ (the limit for the number of iterations of the evolution process), and $P = 5$ (the number of particles in the swarm).

IV. A static LP rounding algorithm for solving the VNF placement subproblem in [1]

In this section, we present a static LP rounding algorithm for solving the VNF placement subproblem in [1]. The algorithm first solves the LP relaxation (LP-II) to obtain a solution $(\boldsymbol{x}^*,\boldsymbol{y}^*,\boldsymbol{r}^*,\boldsymbol{\theta}^*)$. It then uses a static rounding strategy (without updating the LP solution) to map VNFs into cloud nodes taking into account the consistency of the current rounding variable with other already rounded variables. For more details, we refer to Algorithm 5.

V. More numerical results

In this section, we provide more simulation results on the performance of the proposed LP relaxation (LP-II) and the proposed algorithms in [1]. This section is organized as follows: Section V-A describes

Algorithm 3 The algorithm for placing the functions of service k into cloud nodes

```
1: Let $s \leftarrow 1$ be the index of the first function of service $k$ that has not been processed;
2: while $s \leq \ell_k$ do
3:     Compute $\text{Score}_v$ for all $v \in \mathcal{V}$ according Eq. (5);
4:     if $\text{Score}_v = 0$ for all $v \in \mathcal{V}$ then
5:         Stop and declare that the algorithm fails to find a VNF placement strategy for service $k$;
6:     else
7:         Place functions $f_s^k, \ldots, f_{s+t_{v_0}-1}^k$ at cloud node $v_0$, update the remaining node capacity, and set
           $s \leftarrow s + t_{v_0}$ with probability $\frac{\text{Score}_{v_0}}{\sum_{v \in \mathcal{V}} \text{Score}_v}$;
8:     end if
9: end while
```

the procedure to construct the problem instances; Section V-B presents simulation results to compare the performance of solving the natural LP relaxation (LP-I) and the proposed LP relaxation (LP-II); and Section V-C presents simulation results to demonstrate the efficiency and effectiveness of our proposed algorithms over the state-of-the-art approaches in [2], [3], [4], and [7].

A. Construction of Problem Instances

We test all algorithms on the JANOS-US, NOBEL-GERMANY, and POLSKA network topologies taken from the SNDLIB [8]. We use a similar randomly generated procedure as in [1], which is detailed as follows. 20% of the nodes with the largest degree in these networks are selected as the cloud nodes. The cloud nodes' and links' capacities are randomly generated within $[50, 100]$ and $[15, 30]$, respectively. The NFV and communication delays on the cloud nodes and links are randomly generated within $\{3, 4, 5, 6\}$ and $\{2, 4\}$, respectively. For each service k, nodes $S(k)$ and $D(k)$ are randomly chosen from the available network nodes excluding the cloud nodes; SFC $\mathcal{F}(k)$ is a sequence of functions randomly generated from $\{f^1, \ldots, f^5\}$ with $|\mathcal{F}(k)| = 3$; $\lambda_s(k)$'s are the service function rates which are all set to be the same integer value, randomly generated within $[2, 11]$; Θ_k is set to $20 + (3 * \text{dist}_k + \alpha)$ where dist_k is the delay of the shortest path between nodes $S(k)$ and $D(k)$ and α is randomly chosen in $[0, 5]$. The above parameters are carefully chosen to make sure that the constraints in the network slicing problem are neither too tight nor too loose. For each fixed number of services, 100 problem instances are randomly generated and the results presented below are obtained by averaging over these problem instances.

Algorithm 4 The PSO algorithm for solving the network slicing problem

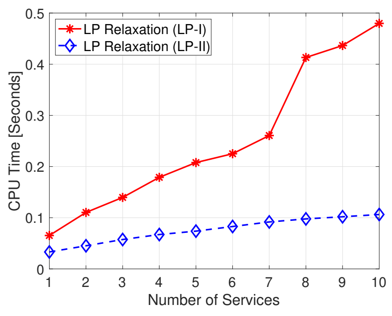
```
1: Sort the services according to their total data rates, i.e., $\sum_{s\in\mathcal{F}(k)} \lambda_s(k)$, $k \in \mathcal{K}$;
2: for each particle $p$ do
3:     for each service $k$ do
4:         Call Algorithm 3 to find a VNF placement solution $\boldsymbol{X}^k(p)$ for service $k$;
5:         $V^k(p)$ is set to 0 or 1 with probability $\bar{r}$ or $1-\bar{r}$, respectively;
6:     end for
7: end for
8: Compute the position $\boldsymbol{B}(p)$ with the best fitness found so far for all $p = 1, \ldots, P$ and the position
   $\boldsymbol{G}$ with the best fitness in the swarm;
9: while $i \leq$ IterLim do
10:    for each particle $p$ do
11:        if $\boldsymbol{X}(p)$ is not a feasible solution for the VNF palcement subproblem then
12:            for each service $k$ do
13:                Call Algorithm 3 to find a VNF placement solution $\boldsymbol{X}^k(p)$;
14:                $V^k(p)$ is set to 0 or 1 with probability $\bar{r}$ or $1-\bar{r}$, respectively;
15:            end for
16:        else
17:            for each service $k$ do
18:                Compute $V^k(p)$ via Eq. (3);
19:                if $V^k(p) = 0$ then
20:                    Call Algorithm 3 to find a VNF placement solution $\boldsymbol{X}^k(p)$ for service $k$;
21:                end if
22:            end for
23:        end if
24:    end for
25:    Recompute the position $\boldsymbol{B}(p)$ with the best fitness for all $p = 1, \ldots, P$ and the position $\boldsymbol{G}$ with
       the best fitness in the swarm;
26:    $i \leftarrow i + 1$;
27: end while
28: If the above procedure finds a feasible solution $\boldsymbol{G}$ for the VNF placement subproblem, solve the
    traffic routing subproblem by solving the multicommodity flow problem.
```

Algorithm 5 A static rounding procedure for solving the VNF placement subproblem

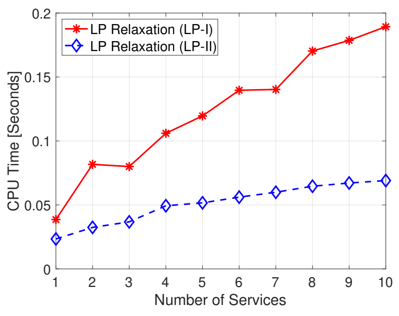
```
1:  Initialize the set $\mathcal{A} = \varnothing$;
2:  Solve problem (LP-II) to obtain its solution $(\boldsymbol{x}^*, \boldsymbol{y}^*, \boldsymbol{r}^*, \boldsymbol{\theta}^*)$ and sort $\{x^*_{v,s}(k)\}$ in the descending
    order;
3:  **for** each $v \in \mathcal{V}$, $k \in \mathcal{K}$, and $s \in \mathcal{F}(k)$ (according to the descending order of $\{x^*_{v,s}(k)\}$) **do**
4:    **if** $x^*_{v,s}(k) = 1$ **then**
5:      Add constraint $x_{v,s}(k) = 1$ into set $\mathcal{A}$ and mark that function $f^k_s$ is assigned;
6:    **else if** function $f^k_s$ has been assigned **then**
7:      Add constraint $x_{v,s}(k) = 0$ into set $\mathcal{A}$;
8:    **else**
9:      Add constraint $x_{v,s}(k) = 1$ into set $\mathcal{A}$;
10:     Add the constraints in set $\mathcal{A}$ into problem (LP-II) to obtain a modified LP;
11:     **if** the modifed LP problem has a feasible $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{r}}, \bar{\boldsymbol{\theta}})$ **then**
12:       Mark that function $f^k_s$ has been assigned;
13:     **else**
14:       Replace constraint $x_{v,s}(k) = 1$ by constraint $x_{v,s}(k) = 0$ in set $\mathcal{A}$;
15:     **end if**
16:   **end if**
17: **end for**
18: **if** all functions $f^k_s$, $k \in \mathcal{K}$, $s \in \mathcal{F}(k)$, are assigned **then**
19:   Declare that $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is a feasible solution for the VNF placement subproblem;
20: **else**
21:   Declare that the algorithm fails to find a feasible solution.
22: **end if**
```

B. Comparison of LP Relaxation (LP-I) and Proposed LP Relaxation (LP-II)

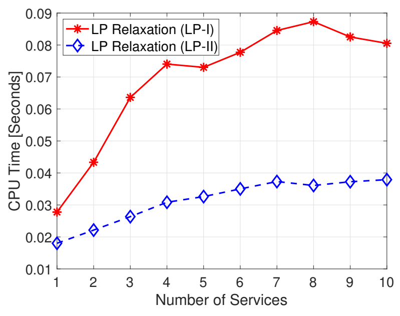
Fig. 3 plots the CPU time versus the numbers of services. From Fig. 3, it can be clearly seen that it is much more efficient to solve relaxation (LP-II) than relaxation (LP-I). In addition, we can observe from Fig. 3 that, with the increasing number of services, the CPU time of (LP-I) generally increases much faster, as compared with that of relaxation (LP-II). This is mainly due to the fact that the numbers of variables and constraints in relaxation (LP-I) generally increase much faster than those in relaxation (LP-II) as the number of services increases.



(a) JANOS-US

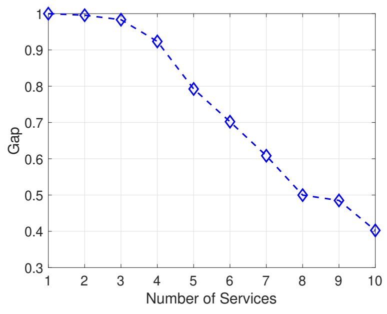


(b) NOBEL-GERMANY

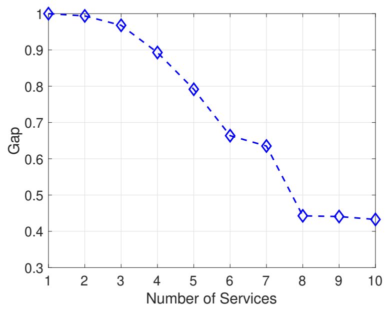


(c) POLSKA

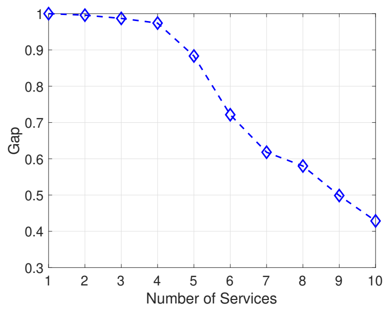
Fig. 3. The CPU time taken by solving relaxations (LP-I) and (LP-II).



(a) JANOS-US



(b) NOBEL-GERMANY



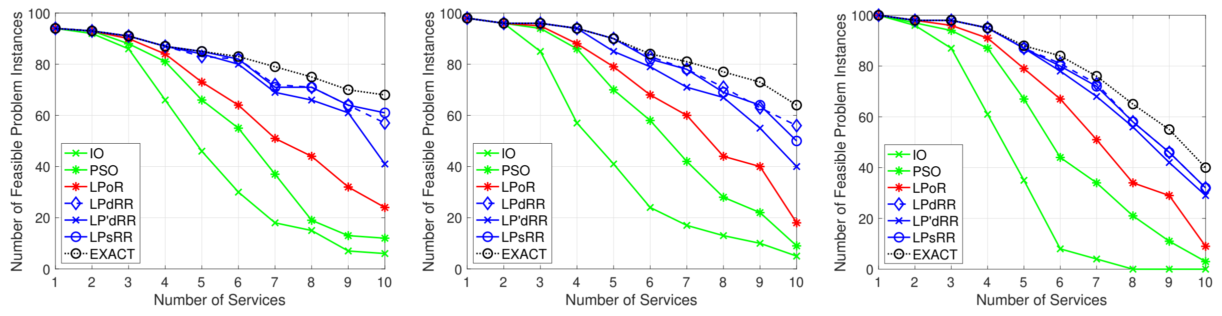
(c) POLSKA

Fig. 4. Average relative gap of the total communication delays.

Fig. 4 plots the average relative gap versus different numbers of services. As observed from Fig. 4, except the case $|\mathcal{K}| = 1$, the relative gap is smaller than 1.0, showing that the LP relaxation (LP-II) is indeed stronger than the LP relaxation (LP-I). In addition, with the increasing number of services, the relative gap becomes smaller. This can be explained as follows. As the number of services increases, the traffic in the network becomes heavier. This further results in the situation that the traffic flows between the two nodes hosting two adjacent functions are likely to transmit over multiple paths. Consequently, the total communication delay returned by solving LP relaxation (LP-I) becomes smaller and relaxation (LP-I) becomes looser.

C. Comparison of Proposed Algorithms and Those in [2], [3], [4], and [7]

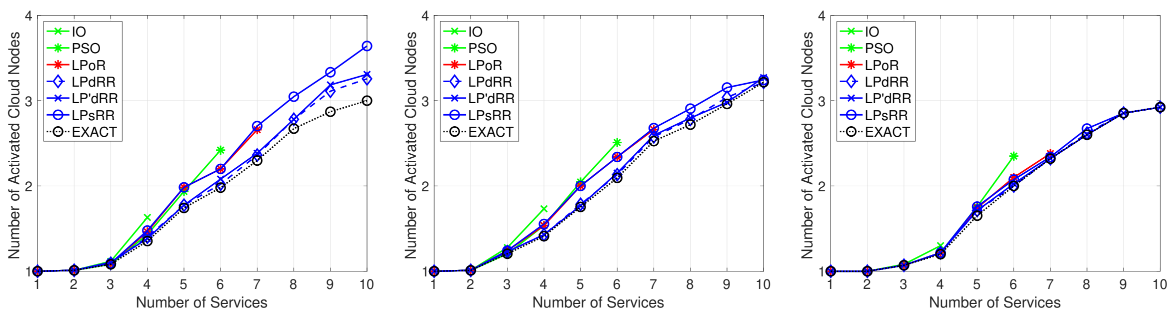
Fig. 5 plots the number of feasible problem instances solved by IO, PSO, LPoR, LPdRR, LP'dRR, LPsRR, and EXACT. First, we can observe from Fig. 5 that compared with LP'dRR, LPdRR can solve a larger number of problem instances, especially when the number of services is large. This clearly shows that the advantage of using the proposed LP relaxation (LP-II) in the proposed LPdRR algorithm, i.e.,



(a) JANOS-US (b) NOBEL-GERMANY (c) POLSKA

Fig. 5. The number of feasible problem instances solved by IO, PSO, LPoR, LPdRR, LP’dRR, LPsRR, and EXACT.

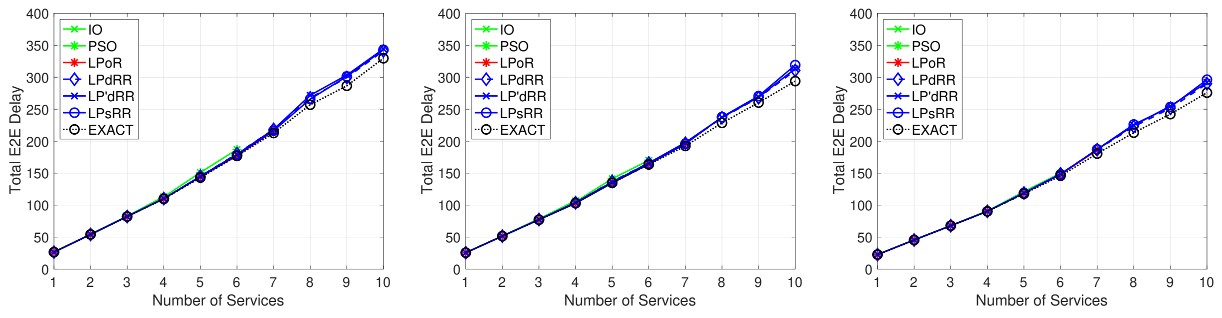
it enables the LPdRR algorithm to find feasible solutions for much more problem instances. Second, the numbers of feasible instances solved by LPdRR and LPsRR are comparable. Finally, we can see the effectiveness of the proposed algorithm LPdRR over IO, PSO, and LPoR in Fig. 5. In particular, as shown in Fig. 5, using the proposed algorithm LPdRR, we can find feasible solutions for much more problem instances, compared with using IO, PSO, and LPoR. Indeed, LPdRR finds feasible solutions for almost all (truly) feasible problem instances (as EXACT is able to find feasible solutions for all (truly) feasible problem instances and the difference of the number of feasible problem instances solved by EXACT and LPdRR is small in Fig. 5).



(a) JANOS-US (b) NOBEL-GERMANY (c) POLSKA

Fig. 6. Average number of activated cloud nodes of IO, PSO, LPoR, LPdRR, LP’dRR, LPsRR, and EXACT.

Fig. 6 plots the average number of activated nodes solved by IO, PSO, LPoR, LPdRR, LP’dRR, LPsRR, and EXACT. As observed from Fig. 5, when $|\mathcal{K}| \geq 5$, $|\mathcal{K}| \geq 7$, and $|\mathcal{K}| \geq 8$, IO, PSO, and LPoR fail to solve a relatively large number of problem instances (generally smaller than 40) and hence we only plot the average number of activated nodes when $|\mathcal{K}| \leq 4$, $|\mathcal{K}| \leq 6$, and $|\mathcal{K}| \leq 7$. From Fig. 6, the number of activated cloud nodes returned by LPdRR or LP’dRR is smaller than that returned by LPoR, IO, PSO, or LPsRR and is comparable with that returned by EXACT.



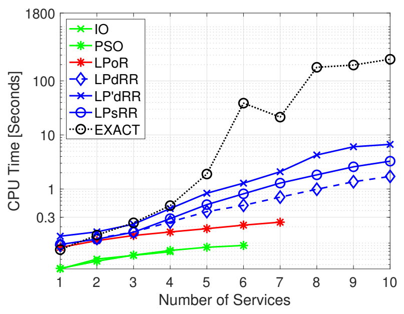
(a) JANOS-US (b) NOBEL-GERMANY (c) POLSKA

Fig. 7. Total E2E delay of IO, PSO, LPoR, LPdRR, LP'dRR, LPsRR, and EXACT.

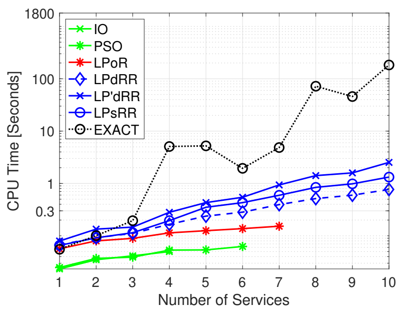
Fig. 7 plots the total E2E delays solved by IO, PSO, LPoR, LPdRR, LP'dRR, LPsRR, and EXACT. Similarly, for IO, PSO, and LPoR, we only plot the total E2E delay when $|\mathcal{K}| \leq 4$, $\mathcal{K} \leq 6$, and $|\mathcal{K}| \leq 7$, respectively. We can observe from Fig. 7 that the E2E delays returned by the seven algorithms are comparable. Overall, (i) the total E2E delays returned by IO and PSO are slightly larger than those returned by LPoR, LPdRR, LP'dRR, LPsRR, and EXACT; and (ii) the E2E delays returned by LPoR, LPdRR, LP'dRR, and LPsRR are slightly larger than that returned by EXACT.

Fig. 8 plots the CPU time of IO, PSO, LPoR, LPdRR, LP'dRR, LPsRR, and EXACT. Similarly, for IO, PSO, and LPoR, we only plot the CPU time when $|\mathcal{K}| \leq 4$, $|\mathcal{K}| \leq 6$, and $|\mathcal{K}| \leq 7$, respectively. First, as expected, LPdRR is much more computationally efficient than LP'dRR and LPsRR, especially when the number of services is large. The efficiency of LPdRR over LP'dRR mainly comes from the fact that solving the proposed LP relaxation is much faster than solving the natural LP relaxation, as demonstrated in Fig. 3, while the efficiency of LPdRR over LP'dRR is due to the fact that the number of solved LPs in LPdRR is much smaller than that solved in LPsRR, as will be illustrated in Fig. 9. Second, compared with EXACT, LPdRR is significantly more efficient. Comparing IO, PSO, LPoR, and LPdRR, IO and PSO are the fastest ones, followed by LPoR. We remark that in most cases, the solution time of IO, PSO, LPoR, and LPdRR is less than one second.

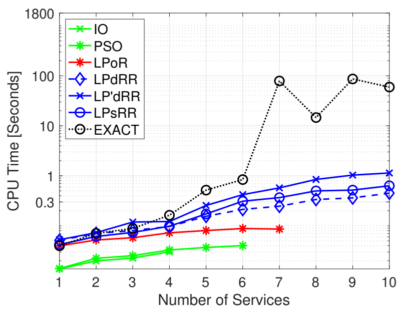
To gain more insight into the computational efficiency of the proposed algorithms LPdRR and LPsRR, we plot the number of solved LPs in these two algorithms in Fig. 9. From this figure, we observe that the number of solved LPs in LPdRR generally increases with the number of services, but is much smaller than its theoretical upper bound in (38) of [1]. This clearly shows that the proposed LPdRR algorithm works well in practice. In addition, compared with LPsRR, LPdRR solves a smaller number of LPs, which is the main reason why LPdRR is faster than LPsRR.



(a) JANOS-US

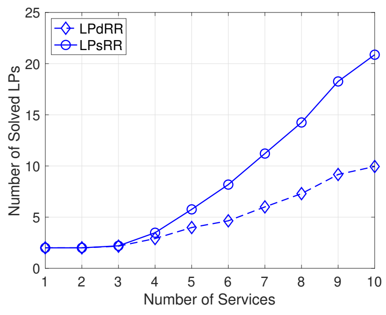


(b) NOBEL-GERMANY

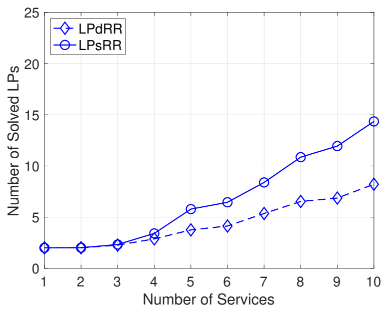


(c) POLSKA

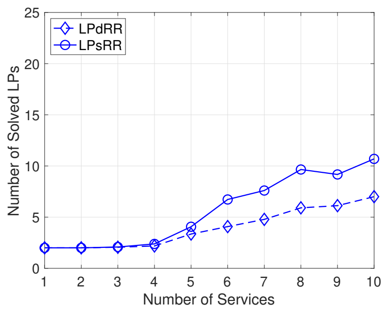
Fig. 8. Average CPU time of IO, PSO, LPoR, LPdRR, LP'dRR, LPsRR, and EXACT.



(a) JANOS-US



(b) NOBEL-GERMANY



(c) POLSKA

Fig. 9. Average number of solved LPs in LPdRR and LPsRR.

References

[1] W.-K. Chen, Y.-F. Liu, F. Liu, Y.-H. Dai, and Z.-Q. Luo. "Towards efficient large-scale network slicing: An dynamic LP rounding-and-refinement approach," 2021. [Online]. Available: https://arxiv.org/abs/2107.14404.

[2] M. Yu, Y, Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: Substrate support for path splitting and migration," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 17-29, April 2008.

[3] Z. Zhang, X. Cheng, S. Su, Y. Wang, K. Shuang, and Y. Luo. "A unified enhanced particle swarm optimization-based virtual network embedding algorithm," *International Journal of Communication Systems*, vol. 26, pp. 1054-1073, January 2012.

[4] M. Chowdhury, M. R. Rahman, and R. Boutaba, "ViNEYard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM Transactions on Networking*, vol. 20, no. 1, pp. 206-219, February 2012.

[5] M. R. Garey and D. S. Johnson, ""Strong" NP-completeness results: Motivation, examples, and implications," *Journal of the ACM*, vol. 25, no. 3, pp. 499-508, July 1978.

[6] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows*. Upper Saddle River, NJ, USA: Prentice Hall, 1993.

[7] W.-K. Chen, Y.-F. Liu, A. De Domenico, Z.-Q. Luo, and Y.-H. Dai. "Optimal network slicing for service-oriented networks with flexible routing and guaranteed E2E latency," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4337-4352, December 2021.

[8] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski, "SNDlib 1.0–survivable network design library," *Networks*, vol. 55, no. 3, pp. 276-286, May 2010.