J Heuristics (2017) 23:137–164
DOI 10.1007/s10732-017-9332-2

A variable neighborhood search for the network design problem with relays

Yiyong Xiao[1] · Abdullah Konak[2]

Received: 11 October 2015 / Revised: 11 April 2017 / Accepted: 13 April 2017 /
Published online: 24 April 2017

Abstract Given a set of commodities to be routed over a network, the network design problem with relays involves selecting a route for each commodity and determining the location of relays where the commodities must be reprocessed at certain distance intervals. We propose a hybrid approach based on variable neighborhood search. The variable neighborhood algorithm searches for the route for each commodity and the optimal relay locations for a given set of routes are determined by an implicit enumeration algorithm. We show that dynamic programming can be used to determine the optimal relay locations for a single commodity. Dynamic programming is embedded into the implicit enumeration algorithm to solve the relay location problem optimally for multiple commodities. The special structure of the problem is leveraged for computational efficiency. In the variable neighborhood search algorithm, the routes of the current solution are perturbed and reconstructed to generate neighbor solutions using random and greedy construction heuristics. Computational experiments on three sets of problems (80 instances) show that the variable neighborhood search algorithm with optimal relay allocations outperforms all existing algorithms in the literature.

Keywords Network design · Telecommunications networks · Variable neighborhood search · Dynamic programming

✉ Abdullah Konak
auk3@psu.edu

Yiyong Xiao
xiaoyiyong@buaa.edu.cn

[1] School of Reliability and System Engineering, Beihang University, Beijing 100191, China

[2] Information Sciences and Technology, Penn State Berks, Tulpehocken Road, P.O. Box 7009, Reading, PA 19610-6009, USA

1 Introduction

The network design problem with relays (NDPR) is a challenging multi-commodity network design problem with side constraints limiting the distance that a commodity can traverse a network without visiting a special node called *relay*. Given a network and a set of commodities, a path is to be determined to route each commodity from its source node to its target node. In order to continue its journey, a commodity must be reprocessed at certain distance intervals on its route. The reprocessing of a commodity takes place at special nodes, called relays, which must be strategically located on the network such that the total distance on any path segment without a relay node is less than an upper bound λ. Being a combination of the network design and facility location problems, the NDPR and its variants have several application areas such as telecommunications networks and distribution systems. In fiber-optic networks, for example, light-wave signals must be regenerated at certain intervals along their routes in order to overcome attenuation (Winters et al. 1993). In natural gas distribution networks, compressors are used to maintain the proper gas pressure (Andre et al. 2009; Kabirian and Hemmati 2007; Tabkhi et al. 2009). In multi-zone truck dispatching systems, loads are transferred to new trucks or exchange drivers at the dispatching hubs en route to their destinations (Ali et al. 2002; Taylor et al. 2001; Üster and Kewcharoenwong 2011). This strategy ensures the continuous movement of loads. In variants of the problem adopted in transportation networks, relay locations (charging stations) should be located within the range of electrical vehicles (Schneider et al. 2014).

In this paper, a partial hybrid algorithm is proposed to solve the version of the problem that includes three types of decisions as defined by Cabral et al. (2007): (1) deciding which edges to be included in the network, (2) determining a route for each commodity, and (3) allocating relay nodes on the commodity routes. In this general form, the NDPR has been sporadically studied in the literature since it was first defined by Cabral et al. (2007) in the context of the wireless network design. Cabral et al. (2007) proposed a mathematical model where both routing and relay location decisions are expressed by a single type of decision variable, which makes the problem separable in terms of commodities. Because the combinations of all feasible paths and relay locations must be generated as input into the model, Cabral et al. (2007)'s model includes a prohibitively large number of decision variables. To address this challenge, Cabral et al. (2007) introduced a column generation approach where only a subset of all feasible path and relay combinations are considered. Although this method does not guarantee optimality, solutions close to optimality can be found in reasonable CPU times. In addition, Cabral et al. (2007) proposed four different construction heuristics to the problem. The common characteristic of these construction heuristics is that a solution is constructed by adding a commodity to the solution one at a time. These construction heuristics differ depending on the computational complexity to select the commodity to add to the partial solution. Kulturel-Konak and Konak (2008) proposed a network flow-based formulation of the problem and developed a genetic algorithm (GA) combined with local search. Their GA's crossover is a greedy construction heuristic, and solutions are improved by local search operators. Konak (2012) showed that relay locations could be optimally determined for a given set of paths by solving a set-covering problem and developed a hybrid-GA where routes are searched by the GA, and relay locations are determined by solving a set-covering problem. Later on, Konak (2014) applied the same principle to solve the two-edge disjoint NDPR where the relay locations are determined by a Lagrangian heuristic. Kabadurmus and Smith (2015) also studied the two-edge disjoint NDPR by considering edge capacities and proposed a three-stage heuristic based on the mathematical formulation of the problem. Laporte and Pascoal (2011) developed a labeling algorithm to find the minimum cost path with a feasible relay assignment for a single commodity. The authors also extended their labeling algorithm to solve the bi-criteria version of the problem where the cost of paths and relays are minimized concurrently. Li et al. (2012) introduced a column generation approach for the directed version of the problem. Their algorithm is similar to the approach of Cabral et al. (2007) in the sense that the same formulation and column generation approach are used, but Lin et al. (2014) utilized MIP and a variant of the labeling algorithm of Laporte and Pascoal (2011) to solve pricing sub-problems for each commodity. In addition, Lin et al. (2014) proposed a tabu search (TS) to the problem.

Optimal relay assignment on a given network, which is a sub-problem of the NDPR, has also been studied in the literature as a stand-alone problem, particularly within the context of fiber-optic telecommunications networks. Gouveia et al. (2003) aimed to determine the optimal locations of switching stations in an optical network under two conditions: (1) the distance that a light path is routed between two switching stations is less an upper bound (a technological limit); and (2) a light path can be routed through only a few number of hops (a quality-of-service requirement). Chen et al. (2010) defined the optimal regenerator assignment problem as minimizing the number of regenerators to be located in an optical network under the constraint that each node pair is connected through at least one path such that the total distance of each subsequence of the edges on the path without regenerators placed on its internal nodes is less than an upper bound. They also showed that the optimal regenerator assignment problem is NP-complete. In a follow-up work, Chen et al. (2015) studied a generalized version of the problem where only a subset of node-pairs is expected to be connected. They also devised a branch-and-cut approach based on a node-weighted directed Steiner forest formulation of the problem, which was shown to solve instances with up to 200 nodes to optimality.

As briefly summarized above, a limited number of heuristic approaches have been proposed to the NDPR although the exact approaches are only applicable to solve small-sized problem instances. One of the challenges of developing effective heuristics to the NDPR is the dependency between the routing and relay decisions. If a combinatorial optimization problem includes multiple types of decisions variables, it is usually difficult to develop effective encoding schemes and search operators that can take advantage of the special structure of the problem. In the case of the NDPR, the researchers exerted a great deal of effort to devise effective crossover/mutation operators for GA (Konak et al. 2009; Kulturel-Konak and Konak 2008) or local move operators for TS (Lin et al. 2014) because traditional ones are not viable alternatives due to the dependency between the route and relay decisions. In such cases, the problem can be decomposed into sub-problems, each having its own solution space based on the type of decision variables, and then each sub-solution space is searched by its specialized algorithm. In the literature, this approach is called partial hybrid algorithms (Talbi 2002). In this paper, a new partial hybrid algorithm is proposed to solve the NDPR. The contributions of this paper are as follows:

- We propose an exact algorithm to assign relays for a given set of commodity routes using dynamic programming.
- We develop a label-setting heuristic to solve the NDPR with a single commodity.
- These two algorithms are embedded within a variable neighborhood search (VNS) framework to generate new solutions from a current solution. It should also be noted that VNS is applied to the NDPR for the first time in this paper.

The rest of the paper is organized as follows. In Sect. 2, we present the description of the NDPR and a mathematical programming model. In Sect. 3, two dynamic programming-based algorithms for the relay location problem (RLP) are proposed for the single-commodity and multi-commodity cases. In Sect. 4, we present a label-setting algorithm for the one-commodity NDPR to initialize the VNS algorithm. The VNS algorithm with the exact approach for the RLP is provided in Sect. 5. In Sect. 6, computational experiments are carried out using 40 existing problem instances and 40 new problem instances, and the results are compared to the existing algorithms in the literature. Finally, Sect. 7 concludes the paper.

2 Formulation of the problem

We re-formulate the NDPR based on the undirected models by Kulturel-Konak and Konak (2008) and the directed model by Li et al. (2012). The parameters and decision variables of the model are listed as follows:

V	set of nodes
n	number of nodes, $n = \text{card}(V)$
i, j	index of nodes, i=1,2,…,n
d_{ij}	distance between nodes i and j
c_{ij}	cost for installing an edge between nodes i and j
λ	maximum distance that a commodity can travel without visiting a relay
A	set of arcs (i, j) that satisfy $d_{ij} \leq \lambda$
r_i	cost of locating a relay at node i
H	set of commodities
m	number of commodities, $m = \text{card}(H)$
k	index of commodities, $k = 1, 2, \ldots, m$
h_{ik}	1 if node i is the source node of commodity k; -1 if node i is the target node of commodity k; and 0, otherwise.

The decision variables used in the model are defined as follows:

x_{ij}	binary variable indicating whether arc (i, j) is selected $(x_{ij} = 1)$ or not $(x_{ij} = 0)$
y_i	binary variable indicating whether node i is selected to locate a relay $(y_i = 1)$ or not $(y_i = 0)$
p_{ijk}	binary variable indicating whether arc (i, j) is used by commodity $k (p_{ijk} = 1)$ or not $(p_{ijk} = 0)$

f_{ik} non-negative continuous variable indicating the cumulative travel distance of commodity k when arrived at node i without visiting a relay. If node i is the source node of commodity k or is not visited by commodity k, then f_{ik} is zero; otherwise, it is a positive number.

The objective function is to minimize the total cost, including edge and relay costs. Problem NDPR:

$$\text{Minimize} \sum_{(i,j)\in A, i<j} c_{ij}x_{ij} + \sum_{i\in N} r_i y_i$$

S.t.

$$\sum_{(i,j)\in A} p_{ijk} - \sum_{(j,i)\in A} p_{jik} = h_{ik} \qquad \forall k \in H, i \in V \tag{1}$$

$$\sum_{(j,i)\in A} p_{jik} \le 1 \qquad \forall k \in H, i \in V \tag{2}$$

$$x_{ij} \ge p_{ijk} \qquad \forall k \in H, (i,j) \in A \tag{3}$$

$$x_{ij} = x_{ji} \qquad \forall (i,j) \in A \tag{4}$$

$$f_{jk} - f_{ik} \ge p_{ijk}d_{ij} - \lambda(1 - p_{ijk} + y_i) \qquad \forall k \in H, (i,j) \in A \tag{5}$$

$$\sum_{(i,j)\in A} p_{ijk}d_{ij} \le f_{jk} \le \lambda \sum_{(i,j)\in A} p_{ijk} \qquad \forall k \in H, j \in V \tag{6}$$

$$x_{ij}, p_{ijk}, y_i \in \{0,1\};\ f_{ik} \ge 0 \qquad \forall (i,j) \in A, k \in H \tag{7}$$

Constraints (1) is the standard node-balance constraints for multicommodity network design problems. Constraint (2) ensures that an arc can be used by a commodity at most once. Constraint (2) also eliminates sub-tours in the solutions. Constraint (3) and (4) requires that edge (i, j) is selected if it is used by any commodity. Constraint (5) is used to calculate the cumulative distance traveled by commodity k without visiting a relay node. Constraint (6) guarantees the cumulative travel distance at any node j must be greater than or equal to the distance it has traveled directly from the predecessor to node j and less than or equal to the upper bound λ. Constraint (6) also forces f_{jk} to be zero if node j is not visited by commodity k, which is indicated by $\sum_{(i,j)\in A} p_{ijk} = 0$.

Problem NDPR uses one less type of decision variables than the model presented in Kulturel-Konak and Konak (2008). Additionally, Constraint (2) ensures that a node appears in a commodity route at most once. Constraint (2) also eliminates sub-tours in a commodity route, reducing the degeneracy of the model. For example, route $\{1 \to 2 \to 3 \to 2 \to \ldots 3 \to 2 \to 3 \to 4\}$ has the same cost as that of $\{1 \to 2 \to 3 \to 4\}$. Note that Constraint (5) alone cannot prevent this phenomenon if nodes 2 and 3 are relays.

The NDPR has the property of symmetry on commodity routes. If a commodity can be sent through the route $P(i \to j)$, where i is the source node and j is the destination node, with relay set R_p along in the tour, then a commodity can also be sent back through the reverse route $P'(j \to i)$ with the same relay set R_p. Furthermore, because the network is undirected (i.e., $d_{ij} = d_{ji}$ and $c_{ij} = c_{ji}$), exchanging the source and destination nodes of a commodity (or multiple commodities) will not change the optimal solution.

To solve large-sized NDPR with a heuristic algorithm, the NDPR is partitioned into two sub-problems: (1) the routing problem, which deals with selecting edges to route the commodities from their source nodes to target nodes and (2) the relay location problem (RLP), which deals with placing relays on the given routes. We propose a VNS approach for the routing problem and develop an exact algorithm to solve the RLP for given routes.

3 Exact and heuristic algorithms for the relay location problem

The relay location problem (RLP) is a sub-problem of the NDPR. In this section, two exact algorithms are introduced for the single-route RLP (S-RLP) and multi-route RLP (M-RLP). These algorithms are used within the VNS algorithm proposed in this paper.

3.1 A dynamic programming (DP) method for the single-route RLP (S-RLP)

In this section, a dynamic programming (DP) algorithm is introduced to determine the optimal relay locations for a given single route. The S-RLP can be described as follows. A given commodity route consists of n nodes that are indexed from 1 (the source node) to n (the target node). The distance between node i and its succeeding node i+1 is denoted as d_i such that $d_i \leq \lambda$. Each node i has a relay installation cost of r_i. The goal of the RLP is to find a set of relays with the minimum relay installation cost such that the total distances between the first node and the first relay, between any two consecutive relays, and between the last relay and last node are all less than λ.

The DP algorithm starts from the target node and moves backward toward the source node to determine the optimal objective function values and decision variables. The backward recursive equation of the DP algorithm for the S-RLP is given as follows:

$$f_i = \begin{cases} 0 & i = n \\ r_i + \min\{f_j | j = i+1, ..., \ i^*\}, & i = n-1, n-2, ..., 2 \\ \min\{f_j | j = 2, ..., \ i^*\}, & i = 1 \end{cases},$$

where f_i represents the optimal total relay cost (including the relay cost at node i) for sending the commodity from node i to the target node, and node i^* is the farthest node that the commodity can be routed from node i without visiting a relay in the middle. For each node i, node i^* can be pre-calculated as $i^* = \max\{i' | i' = i, ..., n; \sum_{j=1}^{i'-1} d_{j,j+1} \leq \lambda\}$. Using the backward recursive equation, the DP algorithm calculates f_i from $i = n$ to 1, where f_1 is the optimal total relay cost for the route. The pseudo-code of the DP algorithm is given in Fig. 1. In the pseudo code, j^* represents the best relay location between nodes $i + 1$ and i^*(Lines from 5 to 10). The optimal relay cost f_i is associated with a set of optimal relay nodes, denoted by U_i, which is

```
Function DP ()
1) Let $f_n \leftarrow 0$
2) Let $U_n \leftarrow \{\}$ // set of relay nodes
3) For each $i$ from $n$-1 to 1step by -1 do
4)     Let $Acc\_dis \leftarrow 0$
5)     Let $j^* \leftarrow i+1$
6)     For each $j$ from $i$+1 to $n$ do
7)             Let $Acc\_dis \leftarrow Acc\_dis + d_{j-1,j}$
8)             If ($Acc\_dis > \lambda$) Then break
9)             If ($f_j < f_{j^*}$) Thenlet $j^* \leftarrow j$
10)  End For
11)    Let $f_i \leftarrow r_i + f_{j^*}$;
12)    Let $U_i \leftarrow \{i\} \cup U_{j^*}$
13)  End For
14)  return $f_1$, $U_1$ // the optimal cost and the node set for placing relays
End
```

Fig. 1 Pseudo-code of dynamic programming for the S-RLP

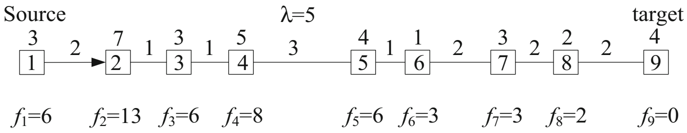

Fig. 2 An example of S-RLP

obtained by $U_i = \{i\} \cup U_{j*}$ (Line 12). The DP algorithm has O(n^2) time complexity. Due to its special structure of the routes, the problem can be effectively solved for a single commodity using dynamic programming.

Figure 2 illustrates an example of the S-RLP, where the relay and edge costs are displayed above the nodes and edges, and the maximum distance λ is 5. The commodity is routed from node 1 to node 9. Table 1 shows the detailed steps of calculating f_i from $i = 9$ to $i = 1$.

3.2 A recursive dynamic programming method for multi-route RLP (M-RLP)

The M-RLP is much more complex than the S-RLP because commodity routes may share paths and relay nodes. Figure 3 illustrates an example of the M-RLP where three commodities share common route segments. Therefore, applying the DP algorithm to the M-RLP one route at a time may yield a good solution but does not guarantee optimality.

Nevertheless, we can apply the DP algorithm within a recursive implicit enumeration framework to obtain the optimal solution for the M-RLP. The algorithm is referred to as the Recursive Dynamic Programming (RDP) algorithm. Let $\pi(k) = \{s_{k1}, s_{k2}, \cdots, s_{kl_k}\}$ represent the route of commodity k such that s_{kp} is the index of the p^{th} node and l_k is the number of the nodes in the route ($p = 1$ for the source node and $p = l_k$ for the target node). We define f_{kp} as the optimal total relay cost of the sub-problem consisting of route k from the p^{th} node to the last node and the remain-

Table 1 The calculation steps of dynamic programming

i	r_i	i^*	Programming for calculating f_i	Opt. j^* $j^* \in [i+1, i^*]$	f_i	Set of relay node $U_i = \{i\} \cup U_{j*}$
9	4	9	$f_9 = 0$		0	
8	2	9	$f_8 = r_8 + \min\{f_9\}$	9	2	8
7	3	9	$f_7 = r_7 + \min\{f_8, f_9\}$	9	3	7
6	1	9	$f_6 = r_6 + \min\{f_7, f_8, f_9\}$	8	3	6, 8
5	4	8	$f_5 = r_5 + \min\{f_6, f_7, f_8\}$	8	6	5, 8
4	5	6	$f_4 = r_4 + \min\{f_5, f_6\}$	6	8	4, 6, 8
3	3	6	$f_3 = r_3 + \min\{f_4, f_5, f_6\}$	6	6	3, 6, 8
2	7	5	$f_2 = r_2 + \min\{f_3, f_4, f_5\}$	5 or 3	13	2, 5, 8 or 2,3,6,8
1	3	4	$f_1 = r_1 + \min\{f_2, f_3, f_4\}$	3	**6**	**1, 3, 6, 8**

The boldface font indicates the optimal solution

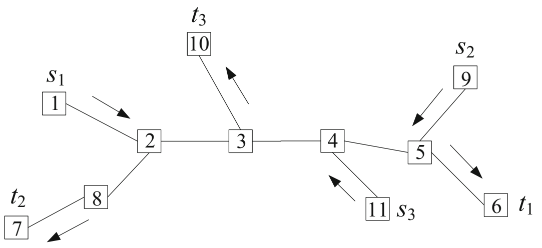

Fig. 3 An example of the M-RLP with three overlapping routes

ing routes from k+1 to m. For example in Fig. 3, $f_{1,1}$ will be the optimal objective value of the original problem, $f_{1,3}$ represents the optimal solution of the sub-problem consisting of routes $\{3 \to 4 \to 5 \to 6\}$, $\{9 \to 5 \to 4 \to 3 \to 2 \to 8 \to 7\}$, and $\{11 \to 4 \to 3 \to 10\}$, and $f_{2,2}$ represents the optimal solution for sub-problem composed of routes $\{5 \to 4 \to 3 \to 2 \to 8 \to 7\}$ and $\{11 \to 4 \to 3 \to 10\}$. Note that f_{kp} includes only the relay cost of newly installed relays for the current sub-problem and excludes the cost of existing relays installed by the higher level of sub-problem (the original problem is the highest level). The recursive dynamic programming equation can be formulated as follows:

$$f_{kp} = \begin{cases} 0 & \forall p = l_m;\, k = m \\ f_{k+1,1} & \forall p = l_k;\, k = m-1, ..., 2, 1 \\ r_i + \min\{f_{kq} | q \in [p+1, p^*]\} & \forall i = S_{kp};\, p = l_k - 1, ..., 2, 1;\, k = m, ..., 2, 1; \end{cases}$$

where p^* is the farthest node that commodity k can be sent to without visiting a relay node if the commodity starts from the p^{th} node. Node p^* can be determined by $p^* = \max\{p' | p < p' \le l_k,\ \sum_{q=p}^{p'-1} d_{s_{kq}, s_{k,q+1}} \le \lambda\}$. The RDP algorithm for M-RLP starts from the first node ($p = 1$) of the first commodity ($k = 1$) and moves to the last node ($p = l_m$) of the last commodity ($k = m$) through a series of recursive calls. Note that the backward recursive procedure used in the DP algorithm for the S-RLP

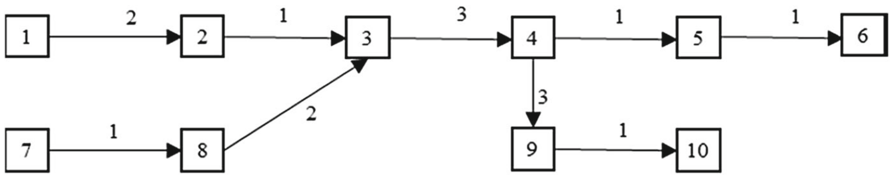

Fig. 4 A numeric example of the M-RLP with $\lambda = 4$

introduced in Sect. 3.1 is not directly applicable to the M-RLP. In the M-RLP, each f_{kp} needs to be calculated multiple times due to sharing of nodes among commodity routes. Figure 4 illustrates this difference and the logic of recursion on an example with two commodity routes, Route 1 [$1 \to 2 \to 3 \to 4 \to 5 \to 6$] and Route 2 [$7 \to 8 \to 3 \to 4 \to 9 \to 10$], such that nodes 3 and 4 are shared by these two routes. The steps of the backward recursive procedure to calculate the values of f_{kp} from f_{26} to f_{11} are given in Table 2. In step 9, f_{15} and f_{16} need to be recalculated after fixing $y_4 = 1$ because placing a relay at node 4 will affect the optimal values for f_{15} and f_{16} (i.e., node 4 is also shared by Route 1). The same rule applies to step 10 because node 3 is also shared by the previous route.

The RDP algorithm is a forward recursive procedure that starts from f_{11} and works toward f_{kp}. The pseudo-code of the RDP algorithm is described in Fig. 5, where the procedure *Main*() is the starting point of recursion and returns the optimal f_{11} as well. The function *Get_f*(k, p) calculates f_{kp} using recursion. A global array is needed to

Table 2 Steps of M-RLP with dynamic programming

step	f_{kp}	First node	Shared	Calculation function
1	f_{26}	10	0	$f_{26} = 0$
2	f_{25}	9	0	$f_{25} = r_9 + \min\{f_{26}\}$
3	f_{24}	4	0	$f_{24} = r_4 + \min\{f_{25}, f_{26}\}$
4	f_{23}	3	0	$f_{23} = r_3 + \min\{f_{24}, f_{25}\}$
5	f_{22}	8	0	$f_{22} = r_2 + \min\{f_{23}, f_{24}\}$
6	f_{21}	7	0	$f_{21} = \min\{f_{22}, f_{23}, f_{24}\}$
7	f_{16}	6	0	$f_{16} = f_{21}$
8	f_{15}	5	0	$f_{15} = r_5 + \min\{f_{16}\}$
9	f_{14}	4	1	$f_{14} = r_4 + \min\{f_{15}, f_{16}\}$. *Note:* f_{15} and f_{16} need to be recalculated with fixing $y_4 = 1$
10	f_{13}	3	1	$f_{13} = r_3 + \min\{f_{14}, f_{15}\}$ *Note:* f_{14} and f_{15} need to be recalculated with fixing $y_3 = 1$
11	f_{12}	2	0	$f_{12} = r_2 + \min\{f_{13}, f_{14}\}$
12	f_{11}	1	0	$f_{12} = \min\{f_{12}, f_{13}\}$

```
1)Main ( )
2)Begin
3)    Let $k \leftarrow 1$, $p \leftarrow 1$
4)    Result $\leftarrow$ $Get\_f(k, p)$
5)End
6)Function $Get\_f(cur\_k, cur\_p)$
7)    For each $k$ from $cur\_k$ to $m$ do
8)      If ($k = cur\_k$) Then let $start\_p \leftarrow cur\_p$
9)      Else let $start\_p \leftarrow 1$
10)     Let $last\_station \leftarrow start\_p$; $Acc\_dis \leftarrow 0$
11)     For each $p$ from $start\_p$+1 to $l_k$ do
12)       Let $i \leftarrow s_{k,p-1}$, $j \leftarrow s_{kp}$, $Acc\_dis \leftarrow Acc\_dis + d_{ij}$
13)       If $Acc\_dis \leq \lambda$ Then
14)         If $y_j$=1 Then let $Acc\_dis \leftarrow 0$, $last\_station \leftarrow p$
15)       Else  //must have a relay in [$last\_station$+1, $p$-1], and find the optimal position $q^*$.
16)         Let $c^* \leftarrow A\_large\_number$
17)         For each $q$ from $last\_station$+1 to $p$-1 do
18)           Let $i' \leftarrow s_{kq}$, $y_{i'} \leftarrow 1$
19)           Let $c \leftarrow r_{i'} + Get\_f(k, q)$
20)           If $c^* > c$ Then Let $c^* \leftarrow c$, $q^* \leftarrow q$
21)           Let $y_{i'} \leftarrow 0$
22)         End For
23)         Return $c^*$
24)       End If
25)     End For
26)   End For
27)   Return 0
28)End
```

Fig. 5 Pseudo-code of the RDP algorithm for multi-route RLP

store the optimal relay position calculated at Line 20 of Fig. 5 (temporary variable q^*). The main functions of the procedure $Get\_f(cur\_k, cur\_p)$ are to find the optimal position q^* between $[last\_station + 1, p - 1]$ (from Line 16 to 22) and to assign a relay to node q^*. During recursion, f_{kp} may be calculated several times as shown in the example above. In each level of recursion, the algorithm stops after the optimal relay station behind the starting node is found. The function $Get\_f(cur\_p, cur\_k)$ returns the optimal f_{cur_p,cur_k} if an optimal q^* exists (Line 23) or returns 0 otherwise (Line 27), which means that the existing assignment of relays is already feasible.

The RDP algorithm is a depth-first-like recursive algorithm, which is known to be P-complete (Reif 1985). In the worst case scenario, the RDP algorithm requires $O(n!)$ recursions. In practice, however, many recursive calls can be avoided by using the following properties.

Property 1 *For a commodity route k of the M-RLP, a relay must be located at node i if* $d_{ji} + d_{ij'} > \lambda$.

Property 1 is straightforward. It simply states that if the total distance of any two consecutive edges of any route is greater than λ, then a relay must be placed at the middle node of the two consecutive edges. According to Property 1, relays can be assigned to some nodes of the routes in advance to reduce the computational burden.

Property 2 *For any route k of the M-RLP, if the commodity can be sent through $P(s \rightarrow t)$, with relay set R_p, then the commodity can also be sent through the reverse route $P'(t \rightarrow s)$ with the same relay set R_p.*

Property 3 *For any two routes k_1 and k_2 of the M-RLP, if route k_1 is a sub-route of or equal to route k_2, then route k_1 can always share the relays of route k_2, without installing any additional relay for route k_1.*

Property 3 indicates that if route k_1 is a part of route k_2, we can remove route k_1 before calling the RDP to optimize the relay cost, and the resulting solution will be the same because the removed route k_1 can always utilize the relays of route k_2.

The above properties can significantly reduce the computational burden of the RDP algorithm in large-sized problems because commodities tend to share route segments. The approach is first to use Property 1 to divide the commodity routes into multiple sub-routes and then use Property 2 to identify those sub-routes that are part of or equal to other routes and remove them (Property 3). After that, we can use the RDP algorithm to optimize the relay cost of remaining sub-routes. For example, given a M-RLP instance with two commodity routes: $[1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 9]$ and $[5 \rightarrow 6 \rightarrow 3 \rightarrow 4 \rightarrow 9 \rightarrow 8]$ such that $d_{23} + d_{34} \geq \lambda$, node 3 must have a relay according to Property 1, and the original two-commodity M-RLP is then divided into a four-commodity M-RLP with routes: $[1 \rightarrow 2 \rightarrow 3]$, $[3 \rightarrow 4 \rightarrow 9]$, $[5 \rightarrow 6 \rightarrow 3]$, and $[3 \rightarrow 4 \rightarrow 9 \rightarrow 8]$. According to Property 3, route $[3 \rightarrow 4 \rightarrow 9]$ is a sub-route of route $[3 \rightarrow 4 \rightarrow 9 \rightarrow 8]$ so it can be removed. As a result, we need only to apply the RDP to the remaining routes $[1 \rightarrow 2 \rightarrow 3]$, $[5 \rightarrow 6 \rightarrow 3]$, and $[3 \rightarrow 4 \rightarrow 9 \rightarrow 8]$ to obtain the optimal relay cost.

4 A label-setting algorithm

In this section, a label-setting heuristic, called $LSA(s, t)$, is introduced to construct a route with feasible relay assignments from a source node s to a target node t such that the total cost of the edges and relays are minimized. The main idea of $LSA(s, t)$ is to construct the route one node at a time by selecting the lowest cost node among a set of candidate nodes in a similar way to Dijkstra's Shortest Path Algorithm. A notable difference is that the optimal cost of the route is calculated by the DP procedure. In Fig. 6, we present the pseudo-code of the $LSA(s, t)$ procedure. In the procedure, v_i denotes the label of node i ($v_i = 0$ for unvisited and $v_i = 1$ for visited), C_i is the cost (edge and relay costs) of a feasible route from node s to node i, and P_i is the predecessor of node i on the route from node s to node i. In the beginning, each node $i \in N$ is labeled as unvisited and assigned to a large temporary cost. In each iteration, C_i is updated for each unvisited node using the DP procedure and the node with the lowest C_i is labeled as visited, i.e. its route from node s is fixed.

The $LSA(s, t)$ procedure is used to repair disconnected routes during the process of creating candidate solutions in the VNS algorithm. In addition, the $LSA(s, t)$ procedure is also used to construct a random feasible initial solution for the VNS algorithm. The procedure to construct a random solution is presented in Fig. 7. In each iteration of the procedure, a commodity k that has not been routed yet is randomly selected, and

```
01) Procedure $LSA(s, t)$
02) For each $i$ in $N$ do
03)   Set $v_i \leftarrow 0$              //initialize the visit states of node $i$ to be unvisited
04)   Set $C_i \leftarrow M$              //initialize the cost from node $s$ to node $i$
05)   Set $P_i \leftarrow s$              //initialize the predecessor of $i$ to be source node $s$
06) End For
07) Set $i \leftarrow s$                  //set $s$ as current node
08) Set $C_i \leftarrow 0$
09) Do while $(i \neq t)$
10)    For each $(i, j)$ in $A$ such that $v_j=0$ do
11)       Generate the path $Q$ from node $s$ to node $j$ by utilizing $P_i$ and arc $(i, j)$
13)       If $(DP(Q)<C_j)$ Then Set $C_j \leftarrow DP(Q)$ and $P_j \leftarrow i$
14)    End For
15)    Set $v_i \leftarrow 1$                          //set node $i$ as visited
16)    Set $i \leftarrow \arg\min \{C_j \mid j \in N \text{ and } v_j=0\}$   //select the next unvisited node
17) End Do
18) Generate a path $Q$ from node $s$ to node $t$ by utilizing $P_t$
19) Set $R \leftarrow DP(Q)$                       //call DP to get the optimal relays of $Q$
20) Return $Q$, $C_t$, and $R$                     //return the route, cost, and relays from node $s$ to node $t$
21) End
```

Fig. 6 Pseudo-code of the LSA(s, t) for the one-commodity NDPR problem

```
1) Procedure Construct_Solution()
2)  Do while($H \neq \{\}$)
3)    Select a commodity $k$ randomly and uniformly from $H$
4)    Set $H \leftarrow H \setminus k$
5)    Call $LSA(s_k, t_k)$ to determine the route ($Q_k$) and relay assignments ($R_k$) for commodity $k$
6)    Set $c_{ij} \leftarrow 0$ for all $(i, j) \in Q_k$
7)    Set $r_i \leftarrow 0$ for all $(i, j) \in R_k$
8)  End Do
9)End
```

Fig. 7 Initial solution construction procedure

the $LSA(s, t)$ procedure is applied to determine the route (Q_k) and relay assignments (R_k) of commodity k. Before constructing the route for the next commodity, the cost of the arcs and relays that are already installed for the previous commodities are set to zero. Thereby, a solution is constructed one commodity at one time in a random order of commodities.

5 Variable neighborhood search (VNS) for the NDPR

Variable neighborhood search (VNS) is a high-level metaheuristic that has been successfully applied to different optimization problems in various fields (Mladenović and Hansen 1997; Hansen and Mladenović 1997; Xiao et al. 2014). In this section, a VNS algorithm is proposed to iteratively improve a current solution that is constructed by the LSA(s, t) procedure. The VNS algorithm removes a randomly selected segment of a randomly selected commodity route from the current solution and constructs new paths to reconnect the disconnected segments. If the resulting solution has a better objective function value, it is accepted as the new current solution; otherwise, it is rejected. We define the neighborhood structure of the current solution in terms of the difference between the numbers of the nodes in the disconnected route and the newly constructed route, i.e., $length(z') - length(z)$ where z and z' denote the selected existing route and the new route, respectively, and $length(z)$ represents the number of the nodes of route z. Thus, given a number K, a new route z' is generated from the current route z as follows:

(i) Two random nodes i and j of the current route z are selected.
(ii) All arcs of the nodes on the path connecting nodes i and j are removed temporarily from the current solution.
(iii) A random new path from node i to node j is found by using a random depth-first search such that $length(z') - length(z) \leq K$.

Parameter K is the neighborhood index that restricts a candidate solution to be generated within the Kth neighborhood of the current solution and limits the extent to which the current solution is allowed to change in the process of generating the candidate solution. The VNS algorithm searches for new solutions in the neighborhood structures by increasing K from zero to K_{max} sequentially. In Fig. 8, we provide an example to illustrate how the neighborhood structures are defined. Figure 8a shows the current route of commodity k starting from source node s_k to target node t_k. In the figure, the solid nodes indicate the route segment to be removed while the arcs between any two empty nodes of the route remain intact. In Fig. 8b–f, the five routes represent candidate solutions generated from different neighborhoods of the current solution.

After the initial solution is set as the current solution, the VNS algorithm first searches for new better solutions with $K = 0$. That means only the candidates with route length shorter than or equal to the current solution will be generated. If a better one is found, then it is accepted as the new current solution (the first improvement

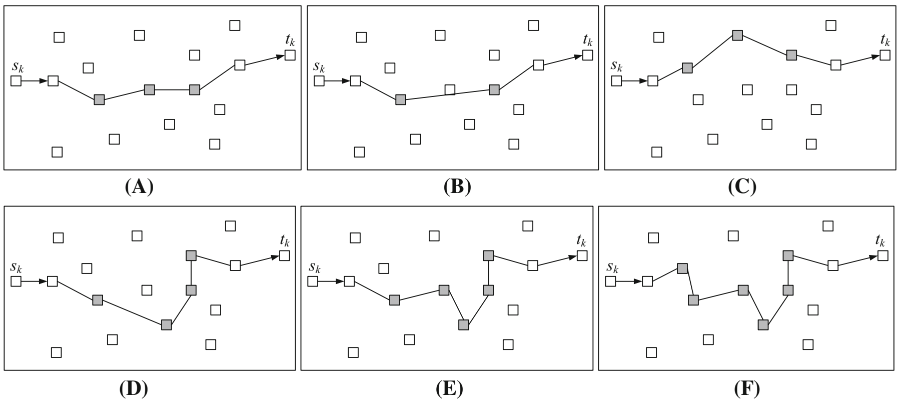

Fig. 8 Examples of neighborhood structures used in the VNS algorithm **a** current solution **b** a new route path with $K = -1$ **c** a new route path with $K = 0$ **d** a new route path with $K = 1$ **e** a new route path with $K = 2$ **f** a new route path with $K = 3$

```
1) Procedure VNS_for_NDPR($P_{\max}$, $K_{\max}$, $N_{\max}$)
2)  Let $P \leftarrow 0$
3)  Do while ($P \le P_{\max}$)
4)    Call $LSA(s, t)$ to initialize a solution with a random commodity order.
5)    Let $f \leftarrow$ the current solution's objective value, $f_{\text{best}} \leftarrow f$
6)    Let $K \leftarrow 0$, $N \leftarrow 0$
7)    Do while ($K \le K_{\max}$)
8)      Select a random commodity $c$, and on its route randomly select two nodes $i$ and $j$
9)      Temporarily remove of all arcs of the nodes between nodes $i$ and $j$
10)     Find a random path connecting nodes $i$ and $j$ such that $length(z')$ - $length(z) \le K$
11)     Call procedure $LSA(s, t)$ to rebuild all other routes that have been broken
12)     Call Recursive Dynamic Programming to optimize relay locations
13)     Calculate the new objective cost $f_{new}$
14)     If ($f_{new} < f$) Then accept the new solution, and let $f \leftarrow f_{new}$, $K \leftarrow 0$, $N \leftarrow 0$
15)                  Else let $N \leftarrow N + 1$
16)     If ($N \ge N_{\max}$) Then let $K \leftarrow K+1$, $N \leftarrow 0$
17)   End Do
18)   If ($f < f_{\text{best}}$) Then let $f_{\text{best}} \leftarrow f$, $P \leftarrow 0$
19)   Else let $P \leftarrow P + 1$
20) End Do
21) Return $f_{\text{best}}$
22) End
```

Fig. 9 Framework of VNS algorithm for NDPR

principle), and the search is restarted by setting $K \leftarrow 0$. If the current solution is not improved after $N_{\max}$ consecutive new solution generations in the neighborhood K, a broader neighborhood is tried by setting $K \leftarrow K + 1$. After the highest level of the neighborhood (i.e, $K = K_{\max}$) is searched without any improvement, the VNS algorithm is restarted with a new initial solution. The search is terminated when either a maximum elapsed CPU time T_{max} has been reached, or the search has been consecutively restarted for $P_{\max}$ times without any improvement in the best solution. In Fig. 9, we present the overall VNS algorithm for the NDPR. The lines (8), (9), and (10) implement the *shaking* concept in VNS, which generates random candidate solutions. Lines (10), (11), and (12) serve as the role of a *greedy local search* procedure that uses procedure $LSA(s, t)$ and the RDP procedures to construct new routes and determine the optimal relays. In other words, the process of generating new candidate solutions (from line (8) to (12)) includes both a random search to discover diverse solutions and a greedy approach to increase the effectiveness of the search.

Note that in line 9 of Fig. 9, the arcs of the removed route segment may also be shared by other commodity routes. Therefore, removing these arcs may disconnect other routes, and these broken routes are reconstructed using the $LSA(s, t)$ procedure. Although the $LSA(s, t)$ procedure is a deterministic greedy algorithm, different solutions can be discovered because the path between the selected nodes i and j is randomly repaired. An example of this process is shown in Fig. 10 where Route 1 (from node s_1 to node t_1) is selected to create a new solution. When the arcs of the nodes on the path between nodes i and j are removed (from Fig. 10a, b), Route 2 is also

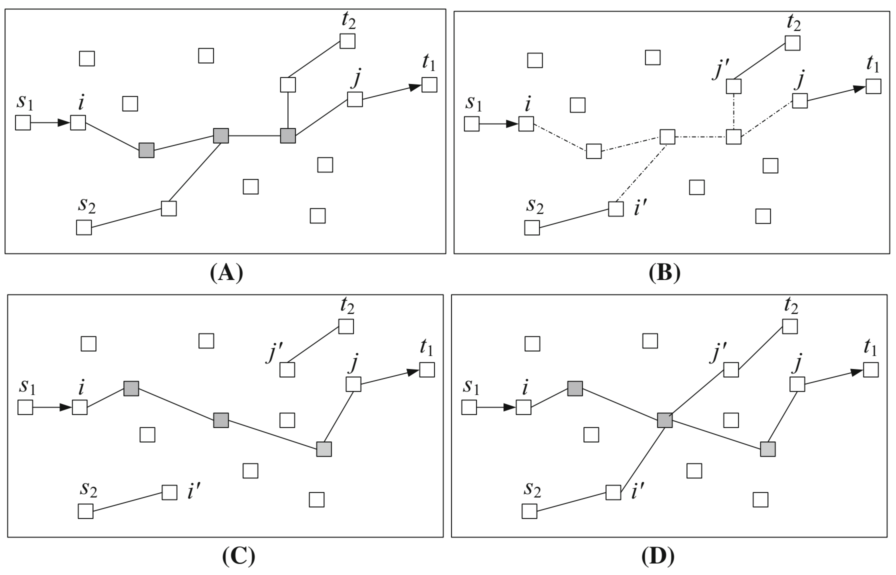

Fig. 10 An example of neighborhood search

broken from node i' to node j'. Therefore, after the selected route is reconnected using a random depth-first search as shown in Fig. 10c, (line 10), the $LSA(s, t)$ procedure is used (line 11) to rebuild any remaining disconnected routes (Fig. 10d).

6 Computational experiments

This section presents the results of computational experiments to test the VNS algorithm and compare its performance with other algorithms from the literature. First, the VNS algorithm is used to solve the problem set defined by Konak (2012). This problem set includes 40 instances of two types—Type-I and Types-II as given in Tables 4 and 6. In Type-I problems, cost c_{ij} and distance d_{ij} of edge (i, j) are defined as the Euclidian distance between nodes i and j.In Type-II problems, the cost is defined as $c_{ij} = \lambda - d_{ij}$ so that there is a strict tradeoff between the edge distance and cost. As shown in Tables 4 and 6, two different levels of λ and $|H|$ values are used for each problem group. In addition, a new set of test problems, referred to as Type-III, are considered in this paper. The Type-III problem set uses the same networks with those of the Type-I and Type-II sets, but cost c_{ij} for edge (i, j) is a random number generated between 0 and λ, and larger numbers of commodities are considered to make the problems more challenging. The parameters of these new test problems are given in Table 8. The VNS algorithm was coded in VC++ 6.0 and run on a PC computer equipped with 3.4GHz Intel® Core$^{\text{TM}}$ i5 3570 CPU and MS Windows 7 system (Fig. 11).

The first set of experiments involved testing the effectiveness of the $LSA(s, t)$ procedure, which is used to initialize solutions and repair disconnected routes while

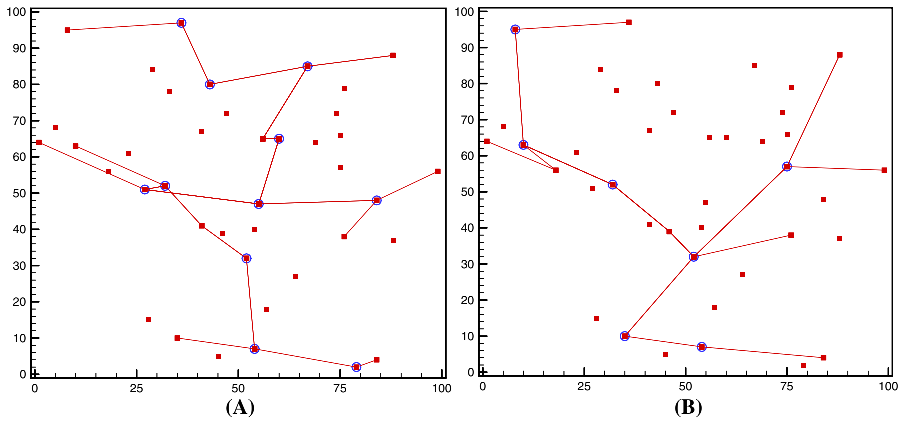

Fig. 11 Two optimal solutions verified by AMPL/CPLEX (*nodes with circle* indicating relays) **a** No. 1 instance of Type-III (cost = 367) **b** No. 2 instance of Type-III (cost = 176)

Table 3 The comparisons of single commodity solutions found by $LSA(s,t)$ to the optimal solutions

Prob. type	Instances	Optimal solutions found by $LSA(s,t)$	Optimality rate (%)	Average deviation from optimal solution (%)
I	150	150	100	0
II	150	150	100	0
III	500	361	72.7	2.1

generating new solutions in the VNS algorithm. The $LSA(s,t)$ procedure was used to find the route and relay assignments for each single commodity of the test problems individually and compared the solutions to the optimal solutions found by Problem NDPR. Note that the optimal route and relay assignments can be determined effectively for a single commodity. In Table 3, the summary of this comparison is presented. The $LSA(s,t)$ procedure found all the optimal solutions of the 300 problem instances of Type-I and Type-II successfully. For Type-III problems, the $LSA(s,t)$ procedure found 361 optimal solutions for the 500 test instances. On the average, the $LSA(s,t)$ procedure yielded 2.1% higher cost than the optimal solutions for 500 problem instances of Type-III.

Table 4 presents the comparative results for the problem instances of Type-I (i.e., $c_{ij} = d_{ij}$). Each problem instance was repeatedly solved for 10 random replications with the same parameter settings of $P_{\max} = 10$, $K_{\max} = 3$, and $N_{\max} = 3000$. In the table, the column *Dev.* indicates the deviation from the best of 10 runs of the VNS algorithm to the previous best-known solutions from Kulturel-Konak and Konak (2008), Konak (2012), or Lin et al. (2014). New best-known solutions were found for instances No. 15, No. 19, and No. 20, and these new solutions are given in Figs. 12, 13 and 14 in "Appendix". The overall performance of the VNS algorithm is compared to the existing algorithms in Table 5, where column *Avg. cost* indicates the average objective function value of 20 Type-I problems, column *Best solutions found*

Table 4 Results of the computational experiment for Type-I problems

Nos.	n	m	λ	$\lvert A\rvert$	Prev. best known	10 runs of VNS			
						Avg	Best	Dev.	Avg time(s)
1	40	5	30	198	**473.80**	**473.80**	**473.80**	0.00	13
2	40	5	35	272	**354.57**	**354.57**	**354.57**	0.00	11
3	40	10	30	198	**518.98**	**518.98**	**518.98**	0.00	37
4	40	10	35	272	**399.76**	400.48	**399.76**	0.00	52
5	50	5	30	279	**283.78**	**283.78**	**283.78**	0.00	9
6	50	5	35	372	**260.23**	**260.23**	**260.23**	0.00	8
7	50	10	30	279	**540.38**	**540.38**	**540.38**	0.00	37
8	50	10	35	372	**407.48**	408.51	**407.48**	0.00	52
9	60	5	30	305	**509.90**	**509.89**	**509.89**	0.00	20
10	60	5	35	412	**377.02**	**377.02**	**377.02**	0.00	20
11	60	10	30	305	**678.84**	**678.84**	**678.84**	0.00	59
12	60	10	35	412	**499.63**	**499.63**	**499.63**	0.00	48
13	80	5	30	641	**356.65**	**356.65**	**356.65**	0.00	63
14	80	5	35	853	**328.80**	**328.80**	**328.80**	0.00	36
15	80	10	30	641	464.99	***463.16***	***463.16***	***−1.83***	93
16	80	10	35	853	**436.75**	**436.75**	**436.75**	0.00	130
17	160	5	30	2773	**287.84**	**287.84**	**287.84**	0.00	52
18	160	5	35	3624	**270.22**	271.47	**270.22**	0.00	71
19	160	10	30	2773	405.64	***404.96***	***404.96***	***−0.68***	432
20	160	10	35	3624	397.59	398.04	***392.24***	***−5.35***	500

The boldfaced-italic and boldfaced fonts indicate new and previous best-known solutions, respectively

shows the number of problems in which the algorithm is able to find the best-known solution, column *Avg. dev. from best* indicates the average value of the deviations (in percentage) of the solutions from their individual new best-known solutions, and *column Avg. time(s)* indicates the average CPU time (in seconds). It can be seen that the VNS algorithm had the best performance (*best of 10*) in terms of *Avg. cost*, *Best solutions found*, and *Avg. dev from best*. The VNS algorithm found the best-known solution for all the problem instances. The second best method is the GA*(avg. of 10)* with the relatively higher computational efficiency, the third one is LSGA*(best of 10)*, and CH1 is the worst one with only one best solution found.

In Tables 6 and 7, the VNS algorithm is compared to the previous methods using Type-II problems (i.e., $c_{ij} = \lambda - d_{ij}$). These results were also found with the same parameters settings of $P_{\max} = 10$, $K_{\max} = 3$, and $N_{\max} = 3000$, and each problem instance was solved for 10 random replications. It can be seen in Table 6 and 7 that although the VNS algorithm did not improve the best-known solutions of the Type-II problems, it is very competitive. Notably, for 19 out of 20 instances (excluding No. 18 instance), the VNS algorithm could always find their previous best-known solutions in each of the 10 replications, indicating a robust performance for Type-II

Table 5 Comparison with the existing algorithms on Type-I problems

Method	Avg. cost	Best solutions found	Avg. dev. from best (%)	Avg. time(s)	CPU type	Method source/solution source
CH1	452.61	1	9.85	–	–	Cabral et al. (2007)
LSGA (best of 10)	414.19	16	0.40	5236	3.2G Intel Xeon	Kulturel-Konak and Konak (2008)
GA (avg. of 10)	419.46	1	1.80	114	3.2G Intel Xeon	Konak (2012)
GA (best of 10)	413.31	10	0.25	1142	3.2G Intel Xeon	Konak (2012)
TS (avg. of 10)	422.28	1	2.21	**70**	PC	Lin et al. (2014)
TS (best of 10)	414.12	9	0.43	240	PC	Lin et al. (2014)
VNS (avg. of 10)	412.69	16	0.12	87	3.4 G Intel Core i5 3570	This paper
VNS (best of 10)	**412.25**	**20**	**0.00**	872	3.4 G Intel Core i5 3570	This paper

The boldfaced fonts indicate best values

Table 6 Results of the computational experiment on 20 instances of Type-II

No.	Node	m	λ	$\|A\|$	Prev. best known	10 runs of VNS			
						Avg	Best	Dev.	Avg time(s)
1	40	5	30	198	**247.27**	**247.27**	**247.27**	0.00	10
2	40	5	35	272	**111.30**	**111.30**	**111.30**	0.00	6
3	40	10	30	198	**292.62**	**292.62**	**292.62**	0.00	17
4	40	10	35	272	**140.51**	**140.51**	**140.51**	0.00	15
5	50	5	30	279	**119.80**	**119.80**	**119.80**	0.00	7
6	50	5	35	372	**155.57**	**155.57**	**155.57**	0.00	9
7	50	10	30	279	**279.70**	**279.70**	**279.70**	0.00	21
8	50	10	35	372	**206.22**	**206.22**	**206.22**	0.00	28
9	60	5	30	305	**317.32**	**317.32**	**317.32**	0.00	18
10	60	5	35	412	**166.35**	**166.35**	**166.35**	0.00	10
11	60	10	30	305	**414.32**	**414.32**	**414.32**	0.00	30
12	60	10	35	412	**242.32**	**242.32**	**242.32**	0.00	20
13	80	5	30	641	**134.73**	**134.73**	**134.73**	0.00	47
14	80	5	35	853	**104.04**	**104.04**	**104.04**	0.00	21
15	80	10	30	641	**187.17**	**187.17**	**187.17**	0.00	65
16	80	10	35	853	**168.62**	**168.62**	**168.62**	0.00	55
17	160	5	30	2773	**78.61**	**78.61**	**78.61**	0.00	78
18	160	5	35	3624	**68.15**	68.19	**68.15**	0.00	71
19	160	10	30	2773	**112.06**	**112.06**	**112.06**	0.00	98
20	160	10	35	3624	**109.12**	**109.12**	**109.12**	0.00	65

The boldfaced fonts indicate previous best-known solutions. The solutions are rounded to two decimal places

problems. The comparison with the existing algorithms is summarized in Table 7. Since Lin et al. (2014) did not solve Type-II problems, the performance of the TS algorithm is not available for comparison. It can be seen that the *VNS (best of 10)* and the *GA (best of 10)* are identical. In terms of the average performance, however, the VNS outperformed the GA for the Type-II problem set.

In Tables 8 and 9, the VNS algorithm and the GA (Konak 2012) are compared using Type-III problems. The VNS algorithm was run with the same setting of $P_{\max} = 10$, $K_{\max} = 3$, and $N_{\max} = 3000$ for ten random replications. The GA was also run for 2000 generations with the population size of 50 as used in Konak (2012). The best and average solutions of the both algorithms are listed in Table 8. Column *Dev. of GA from VNS* presents the deviations of the results of the VNS algorithm from the GA in terms of the *average solution of 10 runs*, *best solution of 10 runs*, and *average computational time*. In terms of the *best solution of 10 runs*, the VNS algorithm outperformed the GA in 33 problems, found the same solutions with the GA in six problems, and performed worse than the GA only in one problem. Regarding the *average solution of 10 runs*, the VNS algorithm significantly outperformed the GA. Table 9 provides the summary of the comparison between the GA and VNS. The average computational

Table 7 Comparison with existing algorithms on 20 instances of Type-II

Method	Avg. cost	Best solutions found	Avg. dev. (%)	Avg. time(s)	CPU type	Method source/solution source
CH1	190.19	4	4.15	–	–	Cabral et al. (2007), Konak et al. (2009)
LSGA (best of 10)	304.64	0	80.82	3698	3.2G Intel Xeon	Kulturel-Konak and Konak (2008)
GA (avg. of 10)	188.90	7	3.37	103	3.2G Intel Xeon	Konak (2012)
GA (best of 10)	**182.79**	**20**	**0.00**	1031	3.2GIntel Xeon	Konak (2012)
VNS (avg. of 10)	182.79	19	0.00	35	3.4G Intel Core i5 3570	This study
VNS (best of 10)	**182.79**	**20**	**0.00**	346	3.4GIntel Core i5 3570	This study

The boldfaced fonts indicate best values

Table 8 Results of the computational experiment on 40 Type-III problems

No.	Node	m	λ	$\|A\|$	10 runs of VNS			10 runs of GA			Dev. of GA from VNS		
					Avg.	Best	Avg. time(s)	Avg.	Best	Avg. time(s)	Avg.	Best.	Time
1	40	5	30	198	**367.0**	**367**	24	376.8	371	71	−9.8	−4	−47
2	40	5	35	272	**176.0**	**176**	12	180.8	**176**	50	−4.8	0	−38.3
3	40	10	30	198	**394.0**	**394**	54	414.4	**394**	72	−20.4	0	−18.1
4	40	10	35	272	204.3	**204**	48	221.4	207	61	−17.1	−3	−13.3
5	40	15	30	198	**438.0**	**438**	93	446.6	441	130	−8.6	−3	−36.7
6	40	15	35	272	254.3	254	54	263.7	**250**	98	−9.4	4	−43.6
7	40	20	30	198	**447.0**	**447**	108	455.1	448	114	−8.1	−1	−6.3
8	40	20	35	272	284.2	**281**	85	288.7	285	108	−4.5	−4	−22.9
9	50	5	30	198	**176.0**	**176**	14	**176.0**	**176**	44	0	0	−29.9
10	50	5	35	272	**160.0**	**160**	11	**160.0**	**160**	74	0	0	−62.5
11	50	10	30	279	**351.0**	**351**	57	373.5	356	77	−22.5	−5	−19.6
12	50	10	35	372	**231.0**	**231**	50	283.2	236	75	−52.2	−5	−25
13	50	15	30	279	**451.0**	**451**	84	520.0	463	116	−69	−12	−31.6
14	50	15	35	372	**444.2**	**442**	119	508.5	460	168	−64.3	−18	−48.8
15	50	20	30	279	499.7	**495**	150	579.8	517	156	−80.1	−22	−5.5
16	50	20	35	372	**471.0**	**471**	202	552.9	539	140	−81.9	−68	61.6
17	60	5	30	305	**378.0**	**378**	16	417.2	414	63	−39.2	−36	−46.9
18	60	5	35	412	285.5	**285**	41	289.1	286	60	−3.6	−1	−19.4
19	60	10	30	305	539.4	**534**	41	574.4	563	93	−35	−29	−52.2
20	60	10	35	412	**367.0**	**367**	61	374.3	**367**	98	−7.3	0	−36.6

Table 8 continued

No.	Node	m	λ	$\lvert A\rvert$	10 runs of VNS			10 runs of GA			Dev. of GA from VNS		
					Avg.	Best	Avg. time(s)	Avg.	Best	Avg. time(s)	Avg.	Best.	Time
21	60	15	30	305	609.5	**607**	120	633.7	611	163	−24.2	−4	−42.8
22	60	15	35	412	431.2	**427**	355	444.8	438	172	−13.6	−11	182.7
23	60	20	30	305	623.5	**620**	217	648.1	631	232	−24.6	−11	−14.6
24	60	20	35	412	**453.0**	**453**	303	499.0	483	208	−46	−30	94.6
25	80	5	30	641	**188.0**	**188**	47	193.9	**188**	106	−5.9	0	−59.4
26	80	5	35	853	**181.0**	**181**	44	198.9	185	107	−17.9	−4	−62.7
27	80	10	30	641	**238.0**	**238**	81	248.2	239	152	−10.2	−1	−70.6
28	80	10	35	853	239.8	**237**	106	261.7	251	129	−21.9	−14	−23.2
29	80	15	30	641	**276.8**	**276**	166	299.8	286	196	−23	−10	−30.1
30	80	15	35	853	279.0	**277**	190	306.1	296	193	−27.1	−19	−3.3
31	80	20	30	641	296.3	**296**	247	319.9	310	270	−23.6	−14	−22.5
32	80	20	35	853	308.0	**307**	359	333.6	320	264	−25.6	−13	94.9
33	160	5	30	2773	128.4	**120**	104	146.2	136	200	−17.8	−16	−96.1
34	160	5	35	3624	92.2	**92**	99	104.0	94	203	−11.8	−2	−104.4
35	160	10	30	2773	176.8	**176**	179	213.3	186	298	−36.5	−10	−119.2
36	160	10	35	3624	149.3	**147**	305	179.6	158	331	−30.3	−11	−26.2
37	160	15	30	2773	216.4	**216**	1872	313.2	295	455	−96.8	−79	1416.8
38	160	15	35	3624	229.7	**227**	1612	288.1	252	554	−58.4	−25	1057.6
39	160	20	30	2773	260.1	**253**	937	340.2	319	589	−80.1	−66	347.6
40	160	20	35	3624	243.7	**242**	1359	306.5	277	705	−62.8	−35	654.2

The boldfaced fonts indicate the best solutions

Table 9 Summarization of experiments on 40 new instances of Type-III

Method	Avg. cost	Best solutions (%)	Avg. dev. from best (%)	Avg. time(s)	CPU type	Method source
GA (avg. of 10)	343.38	2	11.11	185	3.2G Intel Xeon	Konak (2012)
GA (best of 10)	326.60	7	5.15	1849	3.2G Intel Xeon	Konak (2012)
VNS (avg. of 10)	313.48	17	0.61	251		This study
VNS (best of 10)	312.05	39	0.04	2507		This study

times used by the GA and VNS algorithm can be considered comparable considering hardware differences. It is clear that the VNS algorithm performed much better for Type-III problems than the GA did. The solutions found by the GA were 11.06% worse than those of the VNS algorithm. The VNS algorithm determines the optimal relay assignments for each candidate solution while the GA uses the exact approach only for the best solution and utilizes a construction heuristic to determine the relay assignments for other candidate solutions. Therefore, the VNS algorithm is even more competitive than the GA for the problem instances with a higher number of commodities, for which solving the relay assignment sub-problem is more challenging. As seen in Table 9, the average gap between the solutions of the VNS algorithm and the GA increases significantly as the commodity number increases. The edge costs in Type-III problems are randomly assigned, not correlated to the edge's distance such as in the case of Type-I and Type-II problems. Therefore, the interaction between the route selection and relay assignment becomes much more complex in this problem set. Another difference between the GA and the VNS algorithm is that the GA starts with randomly generated solutions while the VNS algorithm starts with more promising solutions generated by the $LSA(s, t)$ procedure.

In Fig. 11, the optimal solutions of two small-sized problems of Type-III (No. 1 and No. 2) are provided. The optimality of the solutions is verified by using CPLEX 12.1 in 2.5 h for No. 1 and 13 h for No. 2.

Finally, the performance of the VNS algorithm was tested for different values of parameters $K_{\max}$ and $N_{\max}$, and the relationship between CPU times and problem size was investigated. The 40 Type-III problems were solved once with parameter combinations by $P_{\max} = 5$, $K_{\max} = 2$, 3, and 4, and $N_{\max} = 1000$, 2000, 3000, and 4000. The results were analyzed using ANOVA to investigate the impact of these parameter settings on solution quality and computational time. Table 10 summarizes the average objective value of 40 problems (column *AVG obj*) and the average CPU time used (column *AVG time*), as well as their corresponding percent deviations (column *Dev.(%)*) from the mean values obtained in the previous experiments reported in Table 7 with parameters $P_{\max} = 10$, $K_{\max} = 3$, and $N_{\max} = 3000$. It can be observed that solution quality is slightly improved by running the algorithm longer or using a larger range of neighborhood structures at the expense of CPU times. However, the slight improvement observed in the average objective value was not statistically significant ($F = 0.498$, $p = 0.608$ for $K_{\max}$ and $F = 0.732$ and $p = 0.533$ for $N_{\max}$ in ANOVA) while the impact of $K_{\max}$ on CPU times was significant ($F = 4.308$, $p = 0.014$) as expected ($N_{\max}$ was included in the ANOVA model for CPU times). In summary, these results indicate that the VNS algorithm performs robustly within the ranges of the parameters used in this study.

We used several linear and non-linear regression models to investigate the relationship between the CPU time requirement of the VNS algorithm and the problem size parameters m, n, $|A|$, and λ. Although the VNS algorithm's procedures that are used to repair solutions and assign optimal relays depend on dynamic programming that has exponential time complexity, the CPU time requirement seems to be a linear function of $m|A|$. The best regression model included $m|A|$ and λ as dependent variables (with adjusted R^2 of 0.725). The empirical study showed that the CPU time strongly

Table 10 Performances of VNS with different parameters on 40 Type-III problems

$P_{\max}$	$K_{\max}$	$N_{\max}$	*AVG obj*	Dev. (%)	*AVG time*	Dev. (%)
5	2	1000	315.45	0.61	59.25	−76.36
5	2	2000	314.28	0.34	72.50	−71.08
5	2	3000	314.35	0.28	132.00	−47.34
5	2	4000	313.70	0.12	154.35	−38.42
5	3	1000	313.93	0.24	110.40	−55.95
5	3	2000	313.75	0.12	125.68	−49.86
5	3	3000	314.23	0.26	154.08	−38.53
5	3	4000	314.15	0.26	212.25	−15.32
5	4	1000	314.70	0.40	70.70	−71.79
5	4	2000	314.05	0.27	137.35	−45.20
5	4	3000	313.00	−0.05	201.98	−19.42
5	4	4000	313.50	0.07	277.33	10.64
10	3	3000	313.48	0	250.65	0

The data in the last row data are from Table 7

depended on $m|A|$ ($p = 0.000$) and was negatively correlated with λ (the relationship was not statically significant with $p = 0.625$).

7 Conclusions

In this paper, a variable neighborhood search (VNS) algorithm is proposed for the NDPR with an exact algorithm for the relay location problem (RLP). The VNS algorithm systematically searches different neighborhoods of the current solution using a random construction heuristic to optimize the routes of commodities. The dynamic programming (DP) for single-route RLP and the recursive dynamic programming (RDP) for the multi-route RLP work within the framework of the VNS algorithm to optimize the relay assignments. Computational experiments on the three sets of problems have shown that the VNS algorithm is a very efficient algorithm for the NDPR, outperforming all existing heuristics in the literature. The VNS algorithm performed particularly well in Type-III problems with random edge costs and larger numbers of commodities. A label-setting algorithm is also proposed for constructing a commodity route with relay assignments. Experiments show that the label-setting algorithm is effective in finding good starting solutions. With these features, the VNS algorithm can solve large-sized problem instances effectively and efficiently.

The VNS algorithm is a partial hybrid algorithm where a complex problem is first decomposed into sub-problems, and then each sub-problem is solved by a specialized algorithm, and a metaheuristic guides the overall optimization process. This approach is applicable to many real-life problems that involve multiple types of interdependent decision variables such as Time-dependent Vehicle Routing Problem with Recharging/Refueling Stations, Facility Layout Problem, Facility Location and Routing Problems in supply chains. For further research, the proposed algorithms can be applied to the capacitated NDPR and the survivable version of the problem. In this paper, edges and relays are assumed to have infinite capacities. Therefore, the commodities tend to share routes or relays. Considering edges and relays with limited capacities will make the problem more applicable to telecommunications networks in particular.

Acknowledgements This work is partly supported by the National Natural Science Foundation of China under Grant No. 71271009.

Appendix

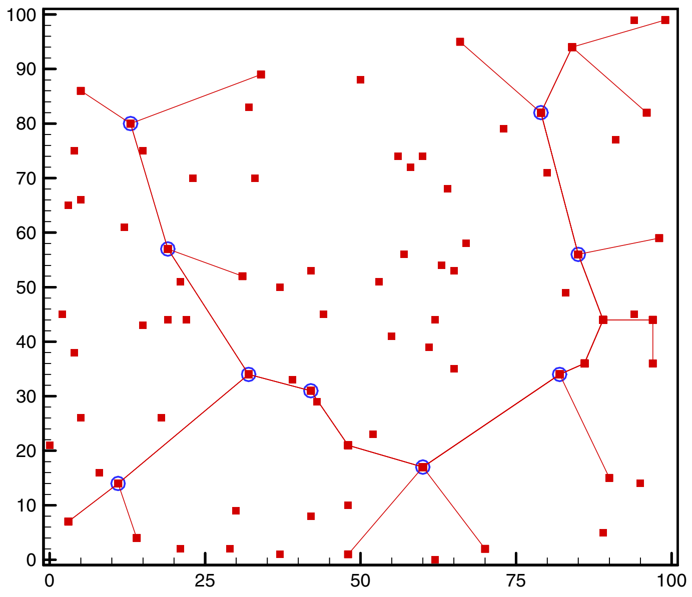

Fig. 12 New best-known solution for No. 15 instance of Type-I (cost = 463.16)

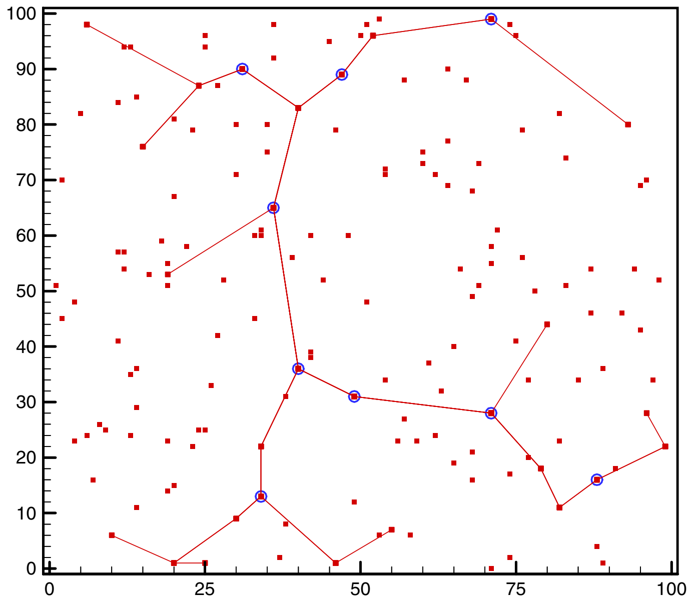

Fig. 13 New best-known solution for No. 19 instance of Type-I (cost = 404.96)

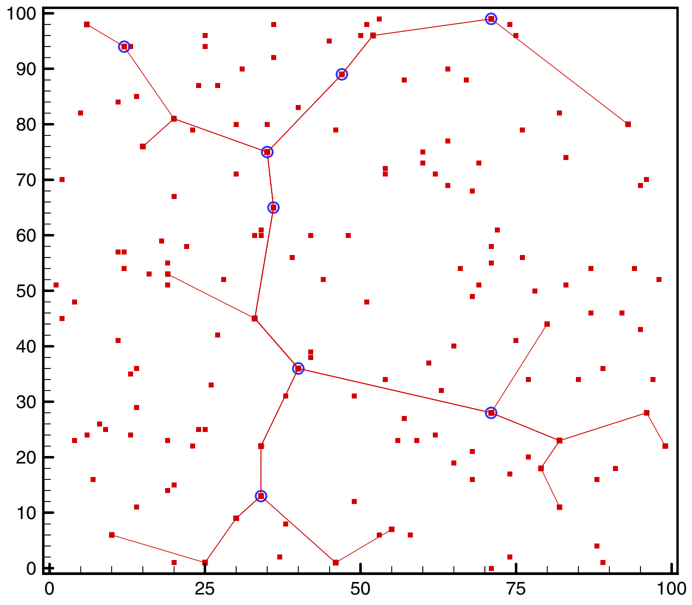

Fig. 14 New best-known solution for No. 20 instance of Type-I (cost = 392.24)

References

Ali, T.H., Radhakrishnan, S., Pulat, S., Gaddipati, N.C.: Relay network design in freight transportation systems. Transp. Res. Part E Logist. Transp. Rev. **38**(6), 405–422 (2002)

Andre, J., Bonnans, F., Cornibert, L.: Optimization of capacity expansion planning for gas transportation networks. Eur. J. Oper. Res. **197**(3), 1019–1027 (2009)

Cabral, E.A., Erkut, E., Laporte, G., Patterson, R.A.: The network design problem with relays. Eur. J. Oper. Res. **180**(2), 834–844 (2007)

Chen, S., Ljubić, I., Raghavan, S.: The regenerator location problem. Networks **55**(3), 205–220 (2010)

Chen, S., Ljubić, I., Raghavan, S.: The generalized regenerator location problem. INFORMS J. Comput. **27**(2), 204–220 (2015)

Gouveia, L., Patricio, P., de Sousa, A.F., Valadas, R.: MPLS over WDM network design with packet level QoS constraints based on ILP models. In: IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies, 30 March–3 April 2003. Piscataway, NJ, USA, IEEE (2003)

Hansen, P., Mladenović, N.: Variable neighborhood search for the p-median. Locat. Sci. **5**(4), 207–226 (1997)

Kabadurmus, O., Smith, A.E.: Multi-commodity *k*-splittable survivable network design problems with relays. Telecommun. Syst. (2015). doi:10.1007/s11235-015-0067-9

Kabirian, A., Hemmati, M.R.: A strategic planning model for natural gas transmission networks. Energy Policy **35**(11), 5656–5670 (2007)

Konak, A.: Network design problem with relays: a genetic algorithm with a path-based crossover and a set covering formulation. Eur. J. Oper. Res. **218**, 829–837 (2012)

Konak, A.: Two-edge disjoint survivable network design problem with relays: a hybrid genetic algorithm and Lagrangian heuristic approach. Eng. Optim. **46**(1), 130–145 (2014)

Konak, A., Kulturel-Konak, S., Smith, A.: Two-edge disjoint survivable network design problem with relays. In: Chinneck, J.W., Kristjansson, B., Saltzman, M. (eds.) Operations Research and Cyber-Infrastructure. Operations Research/Computer Science Interfaces Series, vol. 47, pp. 279–292. Springer, New York (2009)

Kulturel-Konak, S., Konak, A.: A local search hybrid genetic algorithm approach to the network design problem with relay stations. In: Raghavan, S., Golden, B.L., Wasil, E. (eds.) Telecommunications Modeling, Policy, and Technology. Springer, New York (2008)

Laporte, G., Pascoal, M.M.B.: Minimum cost path problems with relays. Comput. Oper. Res. **38**(1), 165–173 (2011)

Li, X., Aneja, Y., Huo, J.: Using branch-and-price approach to solve the directed network design problem with relays. Omega **40**(5), 672–679 (2012)

Lin, S., Li, X., Wei, K., Yue, C.: A tabu search based metaheuristic for the network design problem with relays. In: 2014 11th International Conference on Paper presented at the Service Systems and Service Management (ICSSSM) (2014)

Mladenović, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**(11), 1097–1100 (1997)

Reif, J.H.: Depth-first search is inherently sequential. Inf. Process. Lett. **20**(5), 229–234 (1985)

Schneider, M., Stenger, A., Goeke, D.: The electric vehicle-routing problem with time windows and recharging stations. Transp. Sci. **48**(4), 500–520 (2014)

Tabkhi, F., Pibouleau, L., Azzaro-Pantel, C., Domenech, S.: Total cost minimization of a high-pressure natural gas network. J. Energy Res. Technol. **131**(4), 0430021–04300212 (2009)

Talbi, E.G.: A taxonomy of hybrid metaheuristics. J. Heuristics **8**(5), 541–564 (2002)

Taylor, G.D., Whicker, G.L., Usher, J.S.: Multi-zone dispatching in truckload trucking. Transp. Res. Part E Logist. Transp. Rev. **37**(5), 375–390 (2001)

Üster, H., Kewcharoenwong, P.: Strategic design and analysis of a relay network in truckload transportation. Transp. Sci. **45**(4), 505–523 (2011)

Winters, J.H., Gitlin, R.D., Kasturia, S.: Reducing the effects of transmission impairments in digital fiber optic systems. IEEE Commun. Mag. **31**(6), 68–76 (1993)

Xiao, Y., Zhang, R., Zhao, Q., Kaku, I., Xu, Y.: A variable neighborhood search with an effective local search for uncapacitated multilevel lot-sizing problems. Eur. J. Oper. Res. **235**(1), 102–114 (2014)