Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

Optimisation of transportation service network using κ-node large neighbourhood search

Ruibin Bai[a,*], John R. Woodward[b], Nachiappan Subramanian[c], John Cartlidge[a]

[a] School of Computer Science, Artificial Intelligence and Optimisation Research Group, University of Nottingham Ningbo China, China
[b] School of Computing Science and Mathematics, University of Stirling, Scotland, United Kingdom
[c] University of Sussex, UK

ARTICLE INFO

Article history:
Received 30 April 2015
Revised 6 June 2017
Accepted 7 June 2017
Available online 12 June 2017

Keywords:
Logistics
Transportation network
Service network design
Metaheuristics
Large neighbourhood search

ABSTRACT

The Service Network Design Problem (SNDP) is generally considered as a fundamental problem in transportation logistics and involves the determination of an efficient transportation network and corresponding schedules. The problem is extremely challenging due to the complexity of the constraints and the scale of real-world applications. Therefore, efficient solution methods for this problem are one of the most important research issues in this field. However, current research has mainly focused on various sophisticated high-level search strategies in the form of different local search metaheuristics and their hybrids. Little attention has been paid to novel neighbourhood structures which also play a crucial role in the performance of the algorithm. In this research, we propose a new efficient neighbourhood structure that uses the SNDP constraints to its advantage and more importantly appears to have better reachability than the current ones. The effectiveness of this new neighbourhood is evaluated in a basic Tabu Search (TS) metaheuristic and a basic Guided Local Search (GLS) method. Experimental results based on a set of well-known benchmark instances show that the new neighbourhood performs better than the previous arc-flipping neighbourhood. The performance of the TS metaheuristic based on the proposed neighbourhood is further enhanced through fast neighbourhood search heuristics and hybridisation with other approaches.

1. Introduction

E-commerce and online shopping have rapidly transformed the formats of businesses in recent years. Online shopping companies like `Amazon.com` and China-based `Taobao.com` have seen significant growth in sales in recent years. While most companies are keen to leverage new business opportunities such as online shopping, many of them also encounter new issues, such as providing high quality delivery of billions of products. Hence the problem of logistics has received increasing attention from both industry and the research communities.

Freight transportation has great potential for further improvement in efficiency and service level in the era of big data and cloud computing. The Service Network Design Problem (SNDP) is widely considered as the core problem of freight transportation planning for less-than truck load transport and express deliveries where consolidation is necessary to improve the efficiency. It involves the determination of a cost-effective transportation network and the services which it will provide, while satisfying the constraints related to geographically and temporally diverse demands, network availability, assets capacity, etc. The SNDP is strongly NP-Hard (Ghamlouche et al., 2003) and hence it is impractical to optimally solve the problem of realistic sizes. In fact, the SNDP is generally of large-scale, due to the size of potential network. This is particularly the case when the formulation is based on a time-space network in which each node and each arc has a copy in each period of the scheduling horizon (see Fig. 1).

Various heuristic and metaheuristic approaches have been applied to this problem and substantial progress has been made (Andersen et al., 2011; Bai et al., 2012; Chouman and Crainic, 2014; Crainic et al., 2000; Ghamlouche et al., 2003; 2004; Hoff et al., 2010; Minh et al., 2013; Pedersen et al., 2009). However, almost all of these research studies have focused on various intelligent high-level strategies for better trade-offs between search explorations and exploitations. Here, we consider high-level strategies

* Corresponding author.
E-mail addresses: ruibin.bai@nottingham.edu.cn (R. Bai), john.woodward@stir.ac.uk (J.R. Woodward), N.Subramanian@sussex.ac.uk (N. Subramanian), john.cartlidge@nottingham.edu.cn (J. Cartlidge).

http://dx.doi.org/10.1016/j.cor.2017.06.008

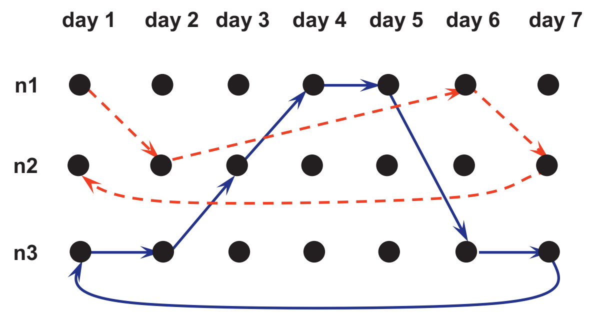


Fig. 1. An example of a time-space network with 3 nodes and 7 periods.

as domain-independent heuristic approaches that do not take specific advantage of a problem's underlying low-level solution structure. Examples of high-level strategies for more efficient search include the tabu-assisted guided local search by Bai et al. (2012) and the hybrid tabu search with path-relinking method by Minh et al. (2013). Analysis of the problem solution structure and its constraints is very limited. As indicated in Kendall et al. (2016), a lot of optimization research studies merely borrow different metaphors without much deep insights on algorithmic or problem properties. These approaches do not satisfy real-world requirements either in terms of solution quality delivered or in computational time required. This is because the SNDP contains some difficult constraints and a flow distribution sub-problem, generally referred to as the Capacitated Multicommodity Min-Cost Flow (CMMCF) problem, which can be very expensive to solve if it is called many times within an iterative metaheuristic approach. This motivates us to develop more efficient metaheuristics for this important and challenging sub-problem. Therefore, unlike the above papers which focus on high-level strategies, in this paper, we propose and study a new larger neighbourhood that exploits the special structure of the SNDP constraints and has much better reachability due to the implicit constraint handling. The experiments on two basic metaheuristic approaches and a hybrid algorithm show that the new neighbourhood is very effective and could be used to develop more efficient algorithms for the SNDP.

The remainder of the paper is structured as follows: Section 2 provides a brief introduction to the SNDP and an overview of the research in freight service network design. Section 3 presents the arc-node based mathematical formulation for SNDP. Section 4 discusses the neighbourhood structure used in the previous studies. Section 5 describes the proposed κ-node neighbourhood operator whose performance is evaluated in Section 6 through a basic Tabu Search (TS) method and a basic Guided Local Search (GLS) method. Section 7 describes a hybrid algorithm based on the κ-node neighbourhood. Section 8 concludes the paper.

2. Literature review

This section provides a brief overview of the previous research into SNDP which is closely related to classic network flow problems (Ahuja et al., 1993). Comprehensive reviews can be found in Crainic (2000), Crainic and Kim (2007) and Wieberneit (2008).

Early work in this field includes Crainic and Rousseau (1986), Powell (1986) and Crainic and Roy (1988). Crainic et al. (1993) applied a TS metaheuristic to the container allocation/positioning problem. Crainic et al. (2000) investigated a hybrid approach for capacitated multicommodity network design (CMND), combining a TS method with pivot-like neighbourhood functions and column generation. Ghamlouche et al. (2003) continued the work and proposed a more efficient cycle-based neighbourhood function for CMND. Experiments with a simple TS framework demonstrated the superiority of the method to the earlier pivot-like neighbourhood functions in Crainic et al. (2000). This approach was later enhanced by adopting a path-relinking mechanism (Ghamlouche et al., 2004).

Barnhart et al. (2002) addressed a real-life air cargo express delivery SNDP. The problem instances are characterised by their large sizes and the addition of further complex constraints to those in the general SNDP model. A tree formulation was introduced and the problem was solved heuristically using a method based on column generation. Armacost et al. (2002) introduced a new mathematical model based on an innovative concept called the *composite variable*, which has a better Linear Programming bound than other models. A column generation method using this new model was able to solve the problem successfully within a reasonable computational time, taking advantage of the specific problem details. However, it may be difficult to generalise their model to other freight transportation applications, especially when there are several classes of services being planned simultaneously. Pedersen et al. (2009) studied more generic SNDP in which a set of *asset balance constraints* was added to model the requirements that the number of incoming vehicles at each node must equal to the outgoing vehicles in order to maintain the continuity of freight services over time. A multi-start metaheuristic, based on TS, was developed and shown to outperform a commercially available MIP solver when computational time was limited to one hour per instance. Andersen et al. (2009) compared the node-arc based formulation, the path-based formulation and a cycle-based formulation for SNDPs. Computational results on a set of small randomly generated instances indicated that the cycle-based formulation gave significantly stronger bounds and hence may allow for much faster solution methods of problems. More recent work by Bai et al. (2012) attempted to further reduce the computational time and investigated a Guided Local Search (GLS) based hybrid approach. The computational study showed that GLS was able to obtain better solutions than Tabu Search (TS) but with less than two thirds of the computational time. However, GLS in that study was based on an arc-flipping neighbourhood which sometimes leads to poor solutions.

Other methods of approaching SNDP have included ant colony and a branch and price method. Barcos et al. (2010) investigated an ant colony optimization approach to address a simplified variant of freight SNDP. The algorithm was able to obtain solutions better than those adopted in the real-world within a reasonable computational time. Andersen et al. (2011) studied a branch and price method for the SNDP. Although the proposed algorithm was able to find solutions of higher quality than the previous methods, the 10-h computational time required by the algorithm poses a great challenge for practical applications.

Variants of SDNP have also been studied. Hoff et al. (2010) investigated a variable neighbourhood search based metaheuristic approach for the service network design with stochastic demand, a problem sharing similar structure to SNDP. However, the neighbourhood functions used in their approach are mainly based on path oriented operators which, like the arc-flipping operator, have limitations in dealing with asset balance constraints. Alumur et al. (2012) studied a heuristic approach for the simultaneous optimisation of hub locations and the service network. A multi-period supply chain network design problem was studied in Carle et al. (2012) and an agent-based metaheuristic was proposed based on the idea of asynchronous cooperation between agents. Nickel et al. (2012) studied a stochastic supply network design problem with financial decisions and risk management for which the authors only managed to solve small instances. Heuristic approaches appear to be the most promising methods for these types of problems. Yaghini et al. (2012) proposed a simulated annealing metaheuristic for the CMND problem without asset-balance constraints. The approach utilised a neighbourhood structure based on the pivoting rules of the Simplex method in order to speed up the search. A multiobjective evolutionary algorithm was proposed for this same problem in Kleeman et al. (2012). However, these metaheuristics do not necessarily perform well on SNDP due to the presence of the asset-balance constraints. Bai et al. (2014) studied a stochastic service network design problem with rerouting. In Bai et al. (2015), a service network design formulation was used to obtain the lower bound of a multi-shift full truckload transportation problem.

It can be seen that the aforementioned research mainly focused on either new models to better capture the complexities of the real-world freight transportation problems or new generic strategies to search the solution space more efficiently. However, limited research has been done to investigate new neighbourhood functions to tackle the difficult constraints and expensive flow distribution sub-problems. The goal of this paper is to address this gap by studying a new neighbourhood structures for SNDP. The effectiveness of the new structure is evaluated in two basic metaheuristic approaches (TS and GLS) and a hybrid method for a set of well-known SNDP benchmark instances.

3. The freight SNDP problem and model

The SNDP is an important tactical/operational freight transportation planning problem. It is of particular interest for less-than truck load transportation and express delivery services, where consolidation of deliveries is widely adopted in order to maximise the utilisation of freight resources (Crainic, 2000). The SNDP involves the search for optimal or near-optimal service characteristics, including the selection of routes and the vehicle types for each route, the service frequency and the delivery timetables, the flow distribution paths for each commodity, the consolidation policies, and the idle vehicle re-positioning, so that legal, social and technical requirements are met (Wieberneit, 2008).

The SNDP differs from the Capacitated Multicommodity Network Design (CMND) problem, a well-known NP-Hard problem, in that it has an additional source of complexity due to the required *balance constraint* for freight assets in order to ensure that vehicle routes are contiguous and that vehicles are in the correct positions after each planning cycle.

The problem of concern in this paper can be formulated in several ways. We used a node-arc based model described in Pedersen et al. (2009) and also present it here for completeness. The list of notation used in the model is given in Table 1.

Let $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ denote a directed graph with nodes $\mathcal{N}$ and arcs $\mathcal{A}$. Let (i, j) denote the arc from node i to node j. Let $\mathcal{K}$ be the set of commodities. For each commodity $k \in \mathcal{K}$, let $o(k)$ and $s(k)$ denote its origin and destination nodes, respectively. Let y_{ij} be Boolean decision variables, where $y_{ij} = 1$ if arc (i, j) is used in the final design and 0 if it is not used. Let x_{ij}^k denote the flow of commodity k on arc (i, j). Let u_{ij} and f_{ij} be the capacity and fixed cost, respectively, for arc (i, j). Finally, let c_{ij}^k denote the variable cost of moving one unit of commodity k along arc (i, j). The SNDP can then be formulated as follows:minimise

$$z(\mathbf{x}, \mathbf{y}) = \sum_{(i,j)\in\mathcal{A}} f_{ij}y_{ij} + \sum_{k\in\mathcal{K}} \sum_{(i,j)\in\mathcal{A}} c_{ij}^k x_{ij}^k \tag{1}$$

subject to

$$\sum_{k\in\mathcal{K}} x_{ij}^k \le u_{ij}y_{ij} \quad \forall (i, j) \in \mathcal{A} \tag{2}$$

$$\sum_{j\in\mathcal{N}^+(i)} x_{ij}^k - \sum_{j\in\mathcal{N}^-(i)} x_{ji}^k = b_i^k, \quad \forall i \in \mathcal{N}, \forall k \in \mathcal{K} \tag{3}$$

$$\sum_{j\in\mathcal{N}^-(i)} y_{ji} - \sum_{j\in\mathcal{N}^+(i)} y_{ij} = 0 \quad \forall i \in \mathcal{N} \tag{4}$$

where $x_{ij}^k \ge 0$ and $y_{ij} \in \{0, 1\}$ are the decision variables. The network capacity constraint (2) ensures that the maximum capacity of arc (i, j) is not violated. The flow conservation constraint (3) ensures that the entire flow of each commodity is delivered to its destination, where $\mathcal{N}^+(i)$ denotes the set of outward neighbours of node i and $\mathcal{N}^-(i)$ the set of inward neighbours. b_i^k is the outward flow of commodity k for node i, so we set $b_i^k = d^k$ if $i = o(k)$, $b_i^k = -d^k$ if $i = s(k)$, and $b_i^k = 0$ otherwise. Constraint (4) is the *asset-balance constraint*, which is missing from the standard CMND formulation, as discussed in Section 2 and which ensures the balance of transportation assets (i.e. vehicles) at the end of each planning period.

For a given design variable vector $\bar{\mathbf{y}} =< \overline{y_{00}}, \dots, \overline{y_{ij}}, \dots >$, the problem becomes one of finding the optimal flow distribution variables. Constraint (4) is no longer relevant and the flow must be zero on all closed arcs, so only open arcs have to be considered in the model. Let $\overline{\mathcal{A}}$ denote the set of open arcs in the design vector $\bar{\mathbf{y}}$ and $\overline{\mathcal{N}}$ be the set of nodes in $\overline{\mathcal{A}}$, then flow distribution variables (x_{ij}^k) for all open arcs $((i, j) \in \overline{\mathcal{A}})$ can be obtained by solving the following CMMCF problem, where $x_{ij}^k \ge 0 \ \forall (i, j) \in \overline{\mathcal{A}},\ k \in K$:minimise

$$\bar{z}(\mathbf{x}) = \sum_{k\in\mathcal{K}} \sum_{(i,j)\in\overline{\mathcal{A}}} c_{ij}^k x_{ij}^k \tag{5}$$

subject to

$$\sum_{k\in\mathcal{K}} x_{ij}^k \le u_{ij} \quad \forall (i, j) \in \overline{\mathcal{A}} \tag{6}$$

$$\sum_{j\in\mathcal{N}^+(i)} x_{ij}^k - \sum_{j\in\mathcal{N}^-(i)} x_{ji}^k = b_i^k, \quad \forall i \in \overline{\mathcal{N}}, \forall k \in \mathcal{K} \tag{7}$$

Constraint (6) ensures the total flow on each open arc in $\overline{\mathcal{A}}$ is no more than its capacity. Constraint (7) is same as the constraint (3) which is the flow conservation constraint.

4. A revisit of previous heuristic approaches

In the previous efforts (Bai et al., 2012; Pedersen et al., 2009), neighbourhood search functions were primarily based on single arc state-flipping (otherwise referred to as arc adding/dropping) with the flow distribution handled separately either heuristically (based on a residual graph) or optimally by solving the corresponding CMMCF problem using an LP solver. Interested readers are referred to Pedersen et al. (2009) for more details of this neighbourhood structure.

However, one drawback of this neighbourhood is the inability to maintain solution feasibility in terms of asset-balance constraints. For a feasible solution satisfying the asset-balance constraints, flipping the state of a single arc will typically generate an infeasible solution (i.e. violating constraint (4)). Let us take a simple network in Fig. 2 as an example. In the current configuration (Fig. 2.(a)), the network consists of 8 open arcs (and 4 closed arcs) and is asset-balanced since, for each node, the number of incoming arcs equals to the number of outgoing arcs. Using the neighbourhood function in Pedersen et al. (2009) and Bai et al. (2012), one could generate 12 neighbouring solutions. Unfortunately none of them is feasible due to asset balance constraint violations. For example, opening arc (1,5) will lead to vehicle imbalance at both nodes 1 and 5. Similarly, closing arc (2,1) will lead to asset-balance constraint violations at nodes 1 and 2. In Pedersen et al. (2009) and Bai et al. (2012), this constraint violation issue was addressed by using a special feasibility-recovery procedure at the end of each local

Table 1
List of notation used in the SNDP model.

Notation	Meaning
$\mathcal{N}$	The set of nodes.
$\mathcal{A}$	The set of arcs in the network.
$\mathcal{G} = (\mathcal{N}, \mathcal{A})$	A directed graph with nodes $\mathcal{N}$ and arcs $\mathcal{A}$.
$(i, j) \in \mathcal{A}$	The arc from node i to j.
u_{ij}	The capacity of arc (i, j).
f_{ij}	The fixed cost of arc (i, j).
$\mathcal{K}$	The set of commodities.
$o(k)$	The origin (source) of commodity $k \in \mathcal{K}$.
$s(k)$	The sink (destination) of commodity k.
d^k	The flow demand of commodity k.
c^k_{ij}	The variable cost for shipping a unit of commodity k on the arc (i, j).
x^k_{ij}	The amount of flow of commodity k on the arc (i, j).
y_{ij}	The network design variables. $y_{ij} = 1$ if arc (i, j) is open and 0 if it is closed.
$\mathbf{x}$	The vector of all flow decision variables, i.e. $\mathbf{x} =< x^0_{00}, \ldots, x^k_{ij}, \ldots >$.
$\mathbf{y}$	The vector of all design variables, i.e. $\mathbf{y} =< y_{00}, \ldots, y_{ij}, \ldots >$.
$\mathcal{N}^+(i)$	The set of outward neighbouring nodes of node i.
$\mathcal{N}^-(i)$	The set of incoming neighbouring nodes of node i.
b^k_i	The outward flow of commodity k. $b^k_i = d^k$ if $i = o(k)$, $b^k_i = -d^k$ if $i = s(k)$ and 0 otherwise.
$z(\mathbf{x}, \mathbf{y})$, $z(s)$	The objective of SNDP model, which represents the sum of the fixed cost and the variable cost for given solution vectors $\mathbf{x}$ and $\mathbf{y}$, or expressed in terms of a potential solution s.
$g(s)$, $g(\mathbf{x}, \mathbf{y})$	The objective function which is actually solved, including a penalty for infeasibility, expressed in terms of a potential solution s or the decision variable component vectors $\mathbf{x}$ and $\mathbf{y}$ of s.

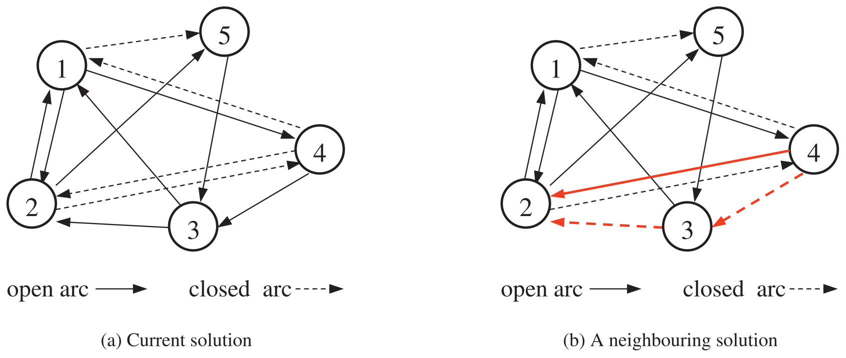


Fig. 2. An illustration of the reachability issue of the arc-flipping neighbourhood. The status of the thicker arcs are changed during the neighbourhood move.

search phase. Although effective in finding a feasible solution, the method may suffer from performance issues when the feasibility-recovery procedure leads to a large increase in costs, and hence inferior solutions.

Another major drawback of the arc-flipping neighbourhood function is the reachability in the search space. Observations from experimental tests in Bai et al. (2012) show that considerable number of neighbourhood moves are rejected during the search and local search methods (both TS and GLS) tend to get stuck at local optima. It appears that this neighbourhood function struggles to reach certain regions of the search space regardless of the number of iterations permitted. This observation explains why the “multiple starts” used in Pedersen et al. (2009) and Bai et al. (2012) is effective. In fact, this can be illustrated by the network in Fig. 2. Assume that the network shown in Fig. 2.(b) is a better feasible solution than Fig. 2.(a). Moving from the solution in Fig. 2.(a) to the solution depicted in Fig. 2.(b) requires closing two arcs 4→ 3 and 3→ 2 and opening arc 4→ 2. Since only one arc can be modified at each neighbourhood move (excluding arcs that are modified during the flow redistribution procedure), in theory it is possible to move to the neighbouring solution in Fig. 2.(b) through 3 successive operators. In practice the success rate of such a move could be extremely low since the first two moves will result in asset imbalance at all three nodes involved and the penalty for this constraint violations can prevent the intermediate solutions from being accepted. In addition, if the flow redistribution during any of these three moves is infeasible, the search will not reach the solution depicted in Fig. 2.(b) from Fig. 2.(a). This explains why the multi-start was required in the previously proposed algorithms.

5. The proposed κ-node neighbourhood

In this section, we describe the proposed new neighbourhood which was originated from the idea of paired-route-flipping. The main purpose is to maintain the feasibility of the solution during the search by changing the status of two carefully selected routes. Each route is a sequence of arcs representing vehicle moves over time. We describe this idea in the following subsection.

5.1. The paired route-flipping

Instead of flipping an arc, we identify a set of arc-flipping operations with automatic feasibility satisfaction in terms of the asset-balance constraint. Fig. 3 illustrates this arc-flipping operator. The solid lines represent open arcs and dotted arcs denote closed arcs. The paired-route-flipping operator involves simultaneously changing the statuses of two routes which share the same source and destination nodes. In this particular example, suppose that the algorithm decides to close a route 1→ 2 → 3. If we can find one of its paired route that also starts at node 1 and finishes at node 3 but with different statuses (i.e. route is closed), the asset balance constraint can be satisfied by simply opening the paired route (i.e. the dotted route). Although this neighbourhood operator can guarantee satisfaction of asset-balance constraints, identifying such a

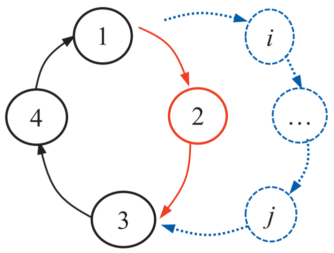


Fig. 3. An illustration of the paired route-flipping neighbourhood. Solid lines are open arcs and dashed ones are closed arcs.

pair of routes is not trivial. On the contrary, it is much easier to focus on nodes rather than arcs, leading to our κ-node neighbourhood structure which we describe in the next subsection.

5.2. The κ-node neighbourhood operator

In this neighbourhood, a subset of κ nodes out of all the nodes are selected and arcs incident upon these nodes are considered for changes. Note that in order to prevent evaluating a candidate solution many times, we require that the change of arcs should involve exactly κ nodes rather than a subset of them. We focus on the small and medium sized neighbourhoods. Large neighbourhoods (e.g. $\kappa > 4$) are not considered since it is impractical to evaluate them within a realistic time limit.

Fig. 4 illustrates the κ-node operator when $\kappa = 2$, 3 and 4, respectively. It is not difficult to see that when $\kappa = 2$, a feasible neighbour may exist only if both arcs connecting the two nodes have a same status (i.e. either both closed or both open). If one of them is open and the other is closed, no feasible neighbouring solution exists.

When $\kappa = 3$, the maximum number of arcs between these nodes is 6. For a feasible current solution s, we denote design variables for arcs $a_0, a_1, \ldots, a_5$ as $y_0, y_1, \ldots, y_5$, respectively. Including the current solution s, the maximum number of neighbouring solutions for a 3-node operator will be $2^6 = 64$. However, not every neighbouring solution will be feasible in terms of asset-balance constraint (4). For any neighbouring solution s', to satisfy asset-balance constraint, the following constraints should be respected, each of which corresponds to one of the three nodes under consideration. We denote the corresponding design variables in s' for arcs $a_0, a_1, \ldots, a_5$ as $y'_0, y'_1, \ldots, y'_5$, respectively.

$$y_0 + y_2 - y_1 - y_3 = y'_0 + y'_2 - y'_1 - y'_3 \tag{8}$$

$$y_1 + y_4 - y_0 - y_5 = y'_1 + y'_4 - y'_0 - y'_5 \tag{9}$$

$$y_3 + y_5 - y_2 - y_4 = y'_3 + y'_5 - y'_2 - y'_4 \tag{10}$$

Condition (8) is obtained from the asset balance constraint for node 0. The left side term is the difference between the number of outgoing and incoming arcs connecting node 0 and the other two nodes in solution s, while the right side term stands for the same difference for node 0 in its neighbouring solution s'. In order to make sure node 0 stays asset-balanced after neighbourhood moves in s', the left side term should be made equal to the right side. That is, any neighbourhood moves should not change the difference between the number of outgoing arcs and incoming arcs for node 0. The same requirements applies to node 1 and node 2, leading to conditions (9) and (10), respectively.

Note that any of the two conditions will be sufficient to ensure feasibility since the third condition can be obtained from the other two conditions. For example, condition (10) can be obtained by simply adding (8) and (9) on both sides correspondingly. In theory, the total possible number of neighbouring solutions of s is $2^\eta - 1$ where η is the number of directed arcs inter-connecting the κ nodes. Hence when $\kappa = 3$, $\eta = 6$, and $2^\eta - 1 = 63$. However, since $y'_0, y'_1, \ldots, y'_5$ take binary values only, these conditions will exclude lots of neighbouring networks that are infeasible. For example, if the left side of condition (8) equals 2 (meaning $y_0 = y_2 = 1, y_1 = y_3 = 0$), none of the 63 neighbours will be feasible because of this condition. If the left side of condition (8) equals 1, i.e. $y'_0 + y'_2 - y'_1 - y'_3 = 1$, including the original network there will be 4 possible feasible neighbouring solutions for this condition. They are: (1, 0, 0, 0), (0, 0, 1, 0), (1, 0, 1, 1), (1, 1, 1, 0). Due to variables y'_4 and y'_5, more solutions are expected if both condition (8) and condition (9) are considered. Nevertheless, the number of asset-balanced neighbouring solutions for s will be significantly smaller than 63.

Similarly when $\kappa = 4$, the following conditions should be satisfied for candidate solutions to ensure the asset-balance at each node:

$$y_0 + y_2 + y_4 - y_1 - y_3 - y_5 = y'_0 + y'_2 + y'_4 - y'_1 - y'_3 - y'_5 \tag{11}$$

$$y_1 + y_6 + y_8 - y_0 - y_7 - y_9 = y'_1 + y'_6 + y'_8 - y'_0 - y'_7 - y'_9 \tag{12}$$

$$y_3 + y_7 + y_{10} - y_2 - y_6 - y_{11} = y'_3 + y'_7 + y'_{10} - y'_2 - y'_6 - y'_{11} \tag{13}$$

$$y_5 + y_9 + y_{11} - y_4 - y_8 - y_{10} = y'_5 + y'_9 + y'_{11} - y'_4 - y'_8 - y'_{10} \tag{14}$$

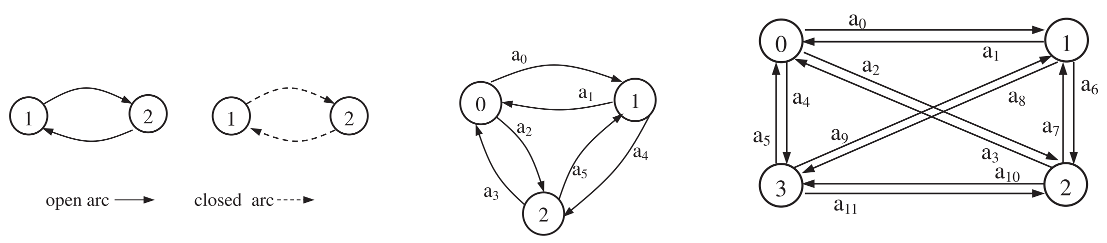
Again, only 3 out the above 4 conditions are active and the other one is redundant. For a medium sized network of 60 nodes, the number of subsets of nodes with cardinality of 4 is $C^4_{60} = 487635$. For each node subset, as mentioned above, the maximum possible number of neighbouring solutions of s is $2^{12} - 1 = 4095$. However, the actual number of feasible neighbouring solutions that satisfy the above conditions is significantly smaller. The size of the neighbourhood depends on the current solution s. For example, there will be no feasible neighbours when the left side of the above conditions takes extreme values (-3 or 3) since it means difference of in-degree and out-degree for all 4 nodes is 3. Any modification of $y_0, \ldots, y_{11}$ will violate at least one of these conditions. The number of feasible neighbours most probably reaches a maximum when the left side of these conditions take values in the middle of permitted range (i.e. equal to 0). That is:

$$y'_0 + y'_2 + y'_4 - y'_1 - y'_3 - y'_5 = 0 \tag{15}$$

$$y'_1 + y'_6 + y'_8 - y'_0 - y'_7 - y'_9 = 0 \tag{16}$$

$$y'_3 + y'_7 + y'_{10} - y'_2 - y'_6 - y'_{11} = 0 \tag{17}$$

Through a binary tree search algorithm, one could solve the above equations and it turns out that only 121 possible feasible neighbours exist as far as the asset-balance constraint is concerned. Despite this reduction, the size of the neighbourhood in a 60-node network when $\kappa = 4$ is still more than 59 million ($121 \times C^4_{60}$). Considering the time taken to solve the flow distribution subproblem for each of these candidate solutions in the neighbourhood, it is impractical to efficiently evaluate neighbourhoods larger than $\kappa = 4$. Even with $\kappa = 4$, it could still be very slow to have a complete evaluation of the neighbourhood. Faster neighbourhood search procedures are required.



(a) $\kappa = 2$ (b) $\kappa = 3$ (c) $\kappa = 4$

Fig. 4. An illustration of κ-node operator when $\kappa = 2, 3,$ and 4.

5.3. Speeding up the neighbourhood search

In this section, we discuss ways that could speed up the neighbourhood search. In the previous neighbourhood structure, there may be solutions which can be discarded directly without ascertaining their objective values. Firstly, given a solution s and one of its neighbouring solutions s', if too many arcs are closed in s' compared to s, there is very little chance that the flow on these arcs can be redistributed among the remaining network. It is therefore not necessary to solve the CMMCF sub-problem. Similarly if a neighbouring solution s' has too many open arcs than the original solution, it is not necessary to evaluate this solution either since the fixed cost would increase dramatically, resulting in a poor solution. These two "extreme" cases are dealt with by adding cut-set inequalities and a heuristic rule respectively which we now discuss.

Let N_κ be the set of κ nodes selected in the κ-node neighbourhood and A_κ be the set of arcs that join any of two nodes from N_κ. For a given κ, the maximum number of arcs incident with these κ nodes is $P_\kappa^2 = \kappa(\kappa - 1)$. For each of node $i \in N_\kappa$, we define the following cut-sets S_i and T_i:

$$S_i = \{N_\kappa \backslash i\}, T_i = \{N \backslash S_i\} \tag{18}$$

Let $CapST_i = \sum_{s\in S_i, t\in T_i} u_{st} y_{st}$ be the aggregated arc capacity from S_i to T_i in a candidate solution with design vector $\mathbf{y}$. Let $DemandST_i$ be the total amount of commodity flows that originate from S_i and destine to T_i. Similarly, let $CapTS_i$ and $DemandTS_i$ be the total available capacity from T_i to S_i and total amount of commodity flows from T_i to S_i, respectively. The necessary conditions for the candidate solution with design variable $\mathbf{y}$ to be feasible are:

$$CapST_i \geq DemandST_i \quad \forall i \in N_\kappa \tag{19}$$

$$CapTS_i \geq DemandTS_i \quad \forall i \in N_\kappa \tag{20}$$

In addition, any modification of arcs related to a node $i \in N_\kappa$ will likely impact on the flows going through its neighbouring nodes. Therefore, similar flow cut-set inequalities can be generated for its neighbouring nodes. Let $S'_i = N^+(i)$ and $T'_i = \{N \backslash S'_i\}, \forall i \in N_\kappa$. We have:

$$CapNST_i \geq DemandNST_i \quad \forall i \in N_\kappa \tag{21}$$

$$CapNTS \geq DemandNTS_i \quad \forall i \in N_\kappa \tag{22}$$

where $CapNST_i = \sum_{s\in S'_i} \sum_{t\in T'_i} u_{st} y_{st}$ is the aggregated arc capacity from S'_i to T'_i for node $\forall i \in N_\kappa$ in solution design vector $\mathbf{y}$, and $CapNTS_i$ is the aggregated arc capacity available from T'_i to S'_i. $DemandNST_i$ and $DemandNTS_i$, respectively, are the aggregated commodity flows from S'_i to T'_i and T'_i to S'_i.

Although useful in avoiding unnecessary CMMCF sub-problem solving, these cut set inequalities can be computationally expensive themselves simply because of the huge number of cuts available. In our implementation, we set the cardinality of the cut set $|S_i| \leq 3$ and we only check against these inequalities for candidate solutions which have 3 arcs or more closed compared to the current solution.

In the case of an "excessive" number of open arcs in s' compared to s, the following condition is used to check whether s' will be evaluated or discarded. Neighbours that do not satisfy this condition will be discarded.

$$\sum_{a\in A_\kappa} f_a \times y'_a \leq \sum_{a\in A_\kappa} f_a \times y_a + w \times \overline{f_\kappa} \tag{23}$$

where $\overline{f_\kappa}$ is the average fixed cost of the arcs in A_κ that are involved in this neighbourhood move. We discard a neighbouring solution if it contains w more open arcs than the original solution, evaluated in terms of the average fixed costs. In our implementation we set $w = 2.5$.

The number of nodes required for κ-node neighbourhood is at least 2. For a given input κ (≥ 2), a neighbouring solution can be generated by making changes to arcs connecting exactly h ($2 \leq h \leq \kappa$) nodes. Therefore, here neighbourhood $\kappa = 3$ will contain neighbouring solutions with changes involved by all possible node pairs and all possible node triplets. The pseudo-code of the neighbourhood is given in Algorithm 1.

For every neighbouring design variable vector $\mathbf{y'}$, the procedure first checks whether asset-balance constraint is respected by this vector. If not, $\mathbf{y'}$ is discarded and the next vector is considered. The asset-balance constraint is checked in the following way. When $h = 2$, as discussed in the previous section, $\mathbf{y'}$ is feasible only if two arcs connecting the two nodes have a same status (i.e. both open or close). When $h = 3$ or $h = 4$, one can check the asset-balance at each node using conditions (8)–(10) and (11)–(14), respectively. When $\kappa > 4$, as we discussed in Section 5.2, the size of the neighbourhood increases significantly. It is impractical to evaluate the entire neighbourhood. Therefore in our experiment, we set $\kappa = 4$. Note that when we generate neighbouring solutions for $h = 3$ or $h = 4$, we should not duplicate neighbours which have been generated for $h = 2$. That is, the neighbourhood moves for $h = 3$ should involve all three nodes, rather than a subset of it. For example, the neighbourhood for node set {1,2,3} should not contain the neighbourhood for node set {1,2}, neither for node set {2,3} or {1,3}. In this way, we can ensure the search starts from smaller neighbourhoods and when we explore larger neighbourhoods, we do not duplicate solution evaluations for previously visited solutions.

Once a neighbouring design variable vector $\mathbf{y'}$ satisfies the asset-balance constraint and the net number of closed arcs is greater or equal to 3, we check it against the inequality conditions (16)–(21) to filter infeasible design variable vectors. After this, the CMMCF procedure is called to find a feasible flow if it exists. If the corresponding node set NS is in the tabu list and aspiration cri-

Algorithm 1 The pseudo-code of the proposed κ-node neighbourhood function with TabuList support. It returns a first-improvement neighbouring solution s^* from the current solution $s = (\mathbf{x}, \mathbf{y})$ as well as the corresponding node set NS^* which defines the neighbourhood. κ is the maximum number of nodes allowed in the κ-node neighbourhood and $z(.)$ is the fitness function.

```
procedure FirstDescent($s, z(.), \kappa$)
    Initialise the best neighbouring solution $s* = \{0\}$, and set $z(s*) = \infty$.
    for $h \leftarrow 2, \kappa$ do
        Generate all possible node sets $N_h$, with each set containing $h$ distinct nodes.
        for all $NS \in N_h$ do
            From the current design variable $\mathbf{y}$ of $s$, generate all its neighbours $\overline{\mathbf{Y}}$ by changing
            the statuses of arcs that interconnect the nodes in $NS$.
            for all $\mathbf{y}' \in \overline{\mathbf{Y}}$ do
                if the asset-balance constraint is violated, skip to the next $\mathbf{y}'$. end if
                if any of inequality constraints (19)-(22) is violated, skip to the next $\mathbf{y}'$. end if
                if CMMCF($\mathbf{y}'$) returns a feasible flow then                ▷ If a feasible flow is found
                    if $NS$ is in TabuList and the aspiration criterion is not met then
                        skip to the next node set $NS$.
                    else Copy the solution to $s'$.
                    end if
                    if $z(s') < z(s)$ then                ▷ A better solution is found, return to caller
                        return $s'$ and $NS$
                    end if
                    if $z(s') < z(s*)$ then
                        $s* = s', NS* = NS$                ▷ Update the best neighbour
                    end if
                end if
            end for
        end for
    end for
    return $s*$ and $NS*$                ▷ Return the best neighbouring solution
end procedure
```

terion is not met, this solution is discarded. Otherwise, it is compared against the initial solution and best solution so far. If a candidate solution improves the initial solution, the procedure returns the first-improved solution. Otherwise, it returns the best solution s^* in the current neighbourhood.

6. Performance evaluation

In this section, we evaluate the performance of the κ-node neighbourhood against two recent metaheuristics based on the arc-flipping neighbourhood. For purposes of comparison, we chose basic TS and basic GLS to avoid complications from other factors such as various intensification and diversification mechanisms.

6.1. A basic TS with κ-node neighbourhood function

We firstly implement a basic TS method with the proposed κ-node neighbourhood function (denoted as TS_κ-node) to evaluate its performance. We compare it against the results reported in Pedersen et al. (2009) by a multi-start TS method given and results reported in Bai et al. (2012) by a tabu assisted multi-start GLS method. Both algorithms use the arc-flipping neighbourhood function. More details and discussions about TS can be found in the book of Glover and Laguna (1997).

The pseudo-code of TS_κ-node algorithm is given in Algorithm 2. The inputs of the algorithm are a feasible initial solution s_0, the objective function of the problem $z(.)$, the maximum number of nodes allowed in the neighbourhood generation κ, and the maximum length of the tabu list TL. Because the neighbourhood search operates on feasible solutions only, the initial solution was generated by the tabu assisted GLS method (TA_MGLS) in Bai et al. (2012) which was stopped as soon as a feasible solution is found. As such, the initial solutions used by

Algorithm 2 A basic TS with κ-node neighbourhood.

```
input An initial feasible solution $s_0$, the objective function $z(.)$, $\kappa$,
  tabu length $TL$.
  Initialise the TabuList, the current solution $s' = s_0$, and the best
  solution $s_b = s_0$.
  while stopping criterion is not met do
      $s', NS \leftarrow$ FirstDescent($s', z(.), k$)        ▷ Get the first-descent
  solution and the node set $NS$
      if $z(s') < z(s_b)$ then
          $s_b = s'$                ▷ Update the best solution
      end if
      TabuList.Add($NS$)      ▷ Add the corresponding node set to the
  TabuList
      if (TabuList.Length $> TL$) then
          TabuList.RemoveFrist                ▷ Maintain the TabuList
      end if
  end while
  return $s_b$
```

the TS method in this experiment are much inferior than the final results reported by TA_MGLS (Bai et al., 2012). In our experiment, we set $\kappa = 4$ to keep the size of the neighbourhood relatively small so that it can be evaluated quickly. We used a fixed length tabu list *TabuList* which is maintained on the first-in-last-out basis. The maximum length is set to $TL = 10$ after some initial tests on a subset of the benchmark instances. Because the proposed κ-node neighbourhood is based on node sets rather than arcs, the tabu list contains the node set which leads to the adoption of the current solution returned by the procedure `FirstDescent(s', z(.), k)`. The procedure repeatedly calls the `FirstDescent(.)` to search for a first-decent neighbouring solution which is not in the tabu list until the stopping criterion is met. In this case, the

Table 2
An initial evaluation of the performance of the proposed κ-node neighbourhood in a basic TS algorithm (TS_κ-node) in comparison with two previous algorithms; TS (Pedersen et al., 2009) and TA_MGLS (Bai et al., 2012). TS was used once only because it was developed into a deterministic algorithm. The best objective values are highlighted in bold.

Instance		TS	TA_MGLS			TS_κ-node		
id	feature	(1 run)	best	avg	worst	best	avg	worst
c37	C20,230,200,V,L	102,919	98,760	99,622	101,606	**97,737**	**98,498**	99,726
c38	C20,230,200,F,L	150,764	142,113	143,867	146,823	**140,146**	**142,770**	146,343
c39	C20,230,200,V,T	103,371	102,137	102,833	104,424	**101,325**	**101,931**	103,001
c40	C20,230,200,F,T	149,942	141,802	143,839	146,141	**140,576**	**141,475**	146,119
c45	C20,300,200,V,L	82,533	79,030	**79,895**	80,888	**78,111**	80,032	81,156
c46	C20,300,200,F,L	128,757	**121,773**	**124,454**	127,607	122,498	124,873	127,039
c47	C20,300,200,V,T	78,571	77,066	**78,302**	80,009	**77,002**	78,393	79,330
c48	C20,300,200,F,T	116,338	**114,465**	**115,836**	117,046	114,886	115,939	117,140
c49	C30,520,100,V,L	55,981	55,732	55,986	56,260	**55,243**	**55,551**	55,995
c50	C30,520,100,F,L	104,533	**100,290**	**102,017**	102,838	101,287	102,838	103,049
c51	C30,520,100,V,T	54,493	54,372	54,708	54,838	**53,759**	**54,177**	54,282
c52	C30,520,100,F,T	105,167	104,574	105,423	106,477	**103,661**	**105,047**	106,018
c53	C30,520,400,V,L	119,735	**116,196**	**116,915**	117,888	116,363	117,638	118,824
c54	C30,520,400,F,L	162,360	**154,941**	**156,008**	157,630	156,506	157,810	160,193
c55	C30,520,400,V,T	120,421	118,336	**118,894**	120,445	**118,253**	119,609	120,594
c56	C30,520,400,F,T	161,978	**157,940**	**159,427**	161,272	158,814	160,096	160,774
c57	C30,700,100,V,L	49,429	49,385	49,457	49,482	**48,826**	**49,210**	49,370
c58	C30,700,100,F,L	63,889	**62,055**	**62,774**	63,397	62,733	62,947	63,200
c59	C30,700,100,V,T	48,202	47,519	47,728	47,937	**47,407**	**47,477**	47,602
c60	C30,700,100,F,T	58,204	**57,571**	58,046	58,447	58,015	**58,015**	58,015
c61	C30,700,400,V,L	103,932	**101,610**	**102,216**	103,008	102,185	102,391	102,827
c62	C30,700,400,F,L	157,043	**142,563**	**144,755**	147,828	142,711	145,397	149,292
c63	C30,700,400,V,T	103,085	**98,657**	**99,726**	100,590	98,926	100,099	101,754
c64	C30,700,400,F,T	141,917	**135,778**	**136,727**	138,004	135,902	137,518	139,666

procedure stops when the maximum allowed time is exhausted. This was set to 2400 s minus the amount of time spent in the initial feasible solution generation phase.

Table 2 presents the computational results by the basic TS with the proposed neighbourhood function (denoted as TS_κ-node) in comparison with two other metaheuristics for this problem; TS (Pedersen et al., 2009) and TA_MGLS (Bai et al., 2012). Since TS in Pedersen et al. (2009), tested on a Pentium IV 2.26 GHz PC with 3600 s CPU time, was developed into a determinstic algorithm, only one run is required. Both TA_MGLS and TS_κ-node were run on a PC with 2.0 GHz Intel Core 2 CPU, single-threaded and a 2400-s time limit in conjunction with Cplex12 as the linear programming solver. Therefore, both TS_MGLS and TS_κ-node uses much less time than TS in Pedersen et al. (2009).

The experiments were based on a set of benchmark instances drawn from Pedersen et al. (2009). This data set consists of 24 instances of different sizes (nodes, arcs, commodities) and distributions of fixed cost, variable cost and capacity. The first three numbers in the instance name represent the number of nodes, the number of arcs and number of commodities respectively. ‘F’ indicates that the fixed cost dominates the cost function while a ‘V’ means dominant variable costs. ‘L’ stands for loose capacity constraints while ‘T’ means capacities are tight. For each instance, 10 independent runs with different random seeds were conducted and their best, mean and worst results are reported. The best results among the three approaches are highlighted in bold. It can be seen that even with a very basic TS method, the new neighbourhood function is able to produce very competitive results. Both the TS method in Pedersen et al. (2009) and the tabu assisted multistart GLS method (TA_MGLS) in Bai et al. (2012) used a multistart framework to diversify the search. It can be seen that the proposed neighbourhood evaluated in a basic TS, performed better than the TS method in Pedersen et al. (2009). It also outperformed TA_MGLS for many instances, particularly small instances. For large instances (e.g. instances with 400 commodities), TS_κ-node was slightly inferior to TA_MGLS. This is probably caused by longer computational time taken by each `FirstDescent(.)` procedure call for larger sized problems which leads to significant increase in CMMCF solution time. A possible improvement for this algorithm is then to develop some faster heuristic flow distribution procedures to reduce the number of CMMCF calls.

6.2. A basic guided local search with new neighbourhood function

We also implemented a basic GLS method with the proposed neighbourhood. The pseudo-code of the algorithm is given in Algorithm 3. GLS is a metaheuristic designed for constraint satisfaction and combinatorial optimisation problems (Voudouris and Tsang, 2003). The underlining idea is to take advantage of information gathered during the search to guide it and enable it to escape local optima. GLS adopts a transformed objective function which includes a penalty to penalise “unattractive” features in a candidate solution. We denote p_r as the current penalty for the presence

Algorithm 3 Pseudo-code for a basic guided local search with new neighbourhood function.

```
input an initial feasible solution $s_0$, an original objective function
    $z(s)$, a set of features $R$, the cost $h_r$ associated with each feature
    $r \in R$ and a scaling parameter $\lambda$.
output an improved solution $s'$.
 1: foreach $r \in R$, set $p_r := 0$
 2: initialise $s \leftarrow s_0$ and $I_r(s)$, set $g(s) = z(s) + \lambda \times \sum_r p_r I_r(s)$
 3: while stopping criterion is not met do
 4:     $s \leftarrow$ FirstDescent$(s, g(s), k)$   ▷ Get the first descent solution
    with regard to $g(s)$
 5:     for all $r \in R$ do
 6:         $util_r(s) = I_r(s) \times \frac{h_r}{1+p_r}$
 7:         Find $r$ with maximum $util_r$, set $p_r = p_r + 1$
 8:     end for
 9: end while
10: return $s' \leftarrow$ best solution found according to the original ob-
    jective function $z(s)$.
```

Table 3
Computational results by a GLS metaheuristic with κ-node neighbourhood (GLS_κ-node) in comparison with results by a basic GLS method with an arc-flipping neighbourhood (GLS) and a multi-start tabu assisted GLS (TA_MGLS) from Bai et al. (2012). All the algorithms were run on a PC with 2.0 GHz Intel Core 2 CPU, single-threaded and a 2400-s time limit in conjunction with Cplex12 as the linear programming solver. The basic GLS in Bai et al. (2012) was run once only due to its deterministic nature and failed to find a feasible solution for 4 instances (denoted as `inf.`). The results by TA_MGLS and GLS_κ-node are based on 10 independent runs. The best results are highlighted in bold.

Instance		GLS	TA_MGLS			GLS_k-node		
id	feature	(1 run)	best	avg	worst	best	avg	worst
c37	C20,230,200,V,L	100,649	98,760	99,622	101,606	**98,395**	**98,567**	98,739
c38	C20,230,200,F,L	145,872	**142,113**	143,867	146,823	142,851	**143,190**	143,529
c39	C20,230,200,V,T	104,863	102,137	**102,833**	104,424	**101,861**	103,010	103,405
c40	C20,230,200,F,T	146,884	**141,802**	**143,839**	146,141	145,463	147,209	148,954
c45	C20,300,200,V,L	80,356	**79,030**	**79,895**	80,888	79,977	80,355	80,918
c46	C20,300,200,F,L	127,356	**121,773**	**124,454**	127,607	125,288	126,474	127,511
c47	C20,300,200,V,T	79,700	**77,066**	**78,302**	80,009	77,807	79,282	80,875
c48	C20,300,200,F,T	131,878	**114,465**	**115,836**	117,046	118,238	118,838	119,715
c49	C30,520,100,V,L	56,166	**55,732**	**55,986**	56,260	56,109	56,137	56,229
c50	C30,520,100,F,L	102,354	**100,290**	**102,017**	102,838	101,942	103,662	105,342
c51	C30,520,100,V,T	inf.	**54,372**	54,708	54,838	54,556	**54,642**	54,664
c52	C30,520,100,F,T	108,223	**104,574**	**105,423**	106,477	105,180	106,833	107,574
c53	C30,520,400,V,L	120,828	**116,196**	**116,915**	117,888	117,420	117,570	117,713
c54	C30,520,400,F,L	162,213	**154,941**	**156,008**	157,630	156,480	157,925	160,347
c55	C30,520,400,V,T	inf.	118,336	**118,894**	120,445	**118,253**	119,470	120,726
c56	C30,520,400,F,T	166,721	**157,940**	**159,427**	161,272	159,113	160,162	161,113
c57	C30,700,100,V,L	49,327	49,385	49,457	49,482	**49,247**	**49,271**	49,367
c58	C30,700,100,F,L	65,270	**62,055**	**62,774**	63,397	62,776	63,503	63,952
c59	C30,700,100,V,T	inf.	**47,519**	**47,728**	47,937	47,704	47,738	47,810
c60	C30,700,100,F,T	58,927	**57,571**	**58,046**	58,447	58,408	58,408	58,408
c61	C30,700,400,V,L	103,317	**101,610**	**102,216**	103,008	102,210	102,356	102,648
c62	C30,700,400,F,L	153,204	**142,563**	**144,755**	147,828	142,711	145,148	149,578
c63	C30,700,400,V,T	inf.	**98,657**	**99,726**	100,590	99,581	100,019	100,380
c64	C30,700,400,F,T	143,447	**135,778**	**136,727**	138,004	135,902	136,795	138,844

of a given feature r in the current solution s, and $I_r(s)$ is an indicator variable such that $I_r(s) = 1$ if the candidate solution s contains feature r and $I_r(s) = 0$ otherwise. h_r is a cost associated with feature r. In this application, we chose all of the arcs as the GLS features and their fixed costs as the feature costs, i.e. $h_r = f_r$ for each arc $r \in \mathcal{A}$. Parameter λ is a scaling coefficient between the original objective function $z(s)$ and the aggregated feature penalties. Since λ is problem instance dependent and is difficult to tune directly, it was estimated by $\lambda = \alpha g(s^*)/\sum_r I_r(s^*)$ where s^* is the current best solution and α is a parameter that is less problem-dependent than λ. At each GLS iteration, the proposed κ-node neighbourhood function `FirstDescent(.)` was used to find a first-descent solution except that the TabuList in `FirstDescent(.)` was set to empty. Therefore, the GLS we tested here is in its basic version.

Similar to the experimental setup in the previous section, the basic GLS metaheuristic started from a feasible solution s_0 generated by the TA_MGLS in Bai et al. (2012) which stops as soon as a feasible solution is found. Similar to TS_κ-node, the stopping criterion was 2400 s of computational time, minus the time spent for generating an initial feasible solution and the size of the neighbourhood is set to $\kappa = 4$. The arcs in the network were chosen to be GLS features R and the fixed cost of each arc is defined as the corresponding feature cost. The GLS parameter was set to $\alpha = 0.05$ based on a preliminary experiment on a number of representative instances.

Table 3 presents the results by GLS with κ-node neighbourhood for the same set of benchmark instances, in comparison with a same basic GLS with an arc-flipping neighbourhood operator (denoted as GLS) and the TA_MGLS in Bai et al. (2012). It can be seen that with a same GLS framework, the proposed κ-node neighbourhood function outperformed the arc-flipping neighbourhood for each of the 24 instances. Compared with TA_MGLS which is much more sophisticated, GLS_κ-node was inferior for most instances but obtained better results for instance C51 and C57, both of which have a small commodity size. GLS_κ-node is generally competitive when the problem size is small. For large instances, each `FirstDescent(.)` call is expensive and hence impedes the search significantly. This is compounded with influence of the transformed objective function $g(s)$ used in GLS that leads to poor solutions since local optima were not reached when the computational time is not sufficient. This also explains why TS_κ-node was able to obtain better solutions than GLS_κ-node in general although both of them started from the same initial solutions and used exactly the same neighbourhood function. Nevertheless, through these two experiments, the new neighbourhood function has shown its effectiveness by producing very promising results, obtaining the new best-known results for many instances. This is largely attributed to its better reachability because of larger neighbourhood sizes and abilities to maintain feasibility. Contrary to many other neighbourhood operators, the proposed new neighbourhood operator uses the constraint violations to their advantage by ignoring lots of infeasible solutions. Compared with the previous neighbourhood function, the superiority of the κ-node operator was demonstrated by the superior results obtained both by the basic GLS and basic TS without the multi-start mechanism which was crucial in a previous hybrid method TA_MGLS in order to prevent the search from getting stuck at local optima.

7. Fast neighbourhood search and hybridisation

In the previous section, we have shown that the proposed neighbourhood showed promising performance when implemented in two basic metaheuristic approaches. It performs better in a tabu search method for small instances. However, the proposed neighbourhood suffers from two main issues: firstly, the size of the neighbourhood is generally very big, therefore a complete neighbourhood evaluation is extremely time consuming. Secondly, evaluation of each candidate solution in a neighbourhood will require solving a CMMCF sub-problem which again is computationally expensive. In this section, we investigate how the proposed neighbourhood method can be improved further. We now describe the heuristics that we used in our experiments to speed up the κ-node neighbourhood search, followed by a hybrid metaheuristic to further enhance the performance of the algorithm.

7.1. Speeding up the neighbourhood search

Due to the size of the neighbourhood and the relatively large solution time for the full CMMCF sub-problem, we adopted the following approximate method which solved a reduced network flow problem. More specifically, in Algorithm 1, line 11, instead of solving the CMMCF sub-problem exactly to obtain the optimal flow for $\mathbf{y}'$, we assume that the existing flows of the current solution s are already well distributed except that the commodity flows through nodes in set NS have to be redistributed since arcs interconnecting these nodes have changed in the neighbourhood move. Let $\mathbf{x}$ be the flow vector before the neighbourhood move and $\mathbf{x}'$ be the flow vector after the neighbourhood move. Let K_r be the set of commodities that have a positive flow through one of nodes in NS. Firstly, we set $\mathbf{x}' = \mathbf{x}$ and then delete from $\mathbf{x}'$ all the flows of every commodity in K_r. Let A_r be the set of open arcs in y' with a positive residual capacity (after the removal of commodity flows for K_r) and rc_{ij} be residual capacity for arc $(i, j) \in A_r$. Finally, let N_r be the set of nodes joined by any of arcs in A_r. The optimal flows for commodities K_r are then obtained by solving the following reduced minimum cost network flow problem.

$$\min \quad z_r = \sum_{k \in K_r} \sum_{(i,j) \in A_r} e_{ij}^k x_{ij}^k \tag{24}$$

subject to

$$\sum_{k \in \mathcal{K}} x_{ij}^k \le rc_{ij} \quad \forall (i, j) \in A_r \tag{25}$$

$$\sum_{j \in \mathcal{N}^+(i)} x_{ij}^k - \sum_{j \in \mathcal{N}^-(i)} x_{ji}^k = b_i^k, \quad \forall i \in N_r, \forall k \in C_r \tag{26}$$

Once the reduced network flow problem is solved, the objective value of s' can be computed through partial evaluation since it is easier to calculate the objective difference between s and s'. In addition, in our implementation, we further speed up the search by only sampling 10% of the neighbourhood randomly when $h = 4$ (see Algorithm 1) since the size of this neighbourhood is extremely large. However, for cases $h = 2$ and $h = 3$, the entire neighbourhoods are evaluated.

7.2. Hybridising with other approaches

In this section we describe how the κ-node neighbourhood based tabu search approach can be hybridised with other approaches to improve its performance. More specifically we hybridise it with a variable fixing heuristic. The proposed hybrid algorithm can be illustrated by Fig. 5.

The hybrid algorithm is divided into four phases, they are MIP_TL, TA_MGLS, TA_k-node, and RMIP. The first stage (denoted as MIP_TL) is the initialisation stage which adopts a Cplex MIP solver to generate a feasible solution by directly solving the SNDP model (1)–(4) within a given time limit (t_1). TA_MGLS is the tabu assisted multi-start guided local search method proposed in Bai et al. (2012). TA_k-node is the tabu search method we described in the previous section. RMIP is a post optimisation procedure which solves a reduced MIP problem based on a pool of elite solutions found during the second and the third phases. The computational

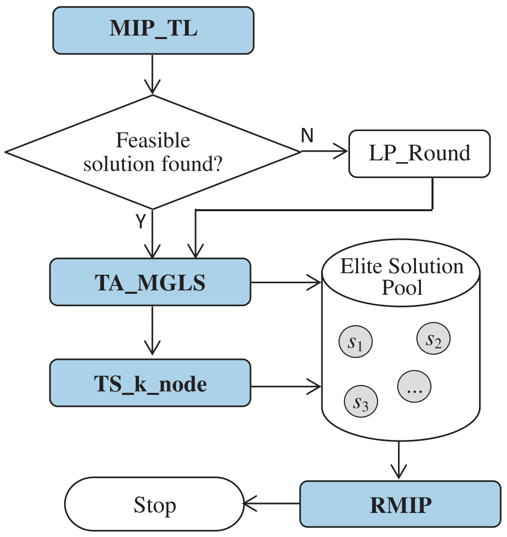


Fig. 5. The diagrammatic illustration of the hybrid algorithm.

time for these three phases are denoted as t_2, t_3 and t_4, respectively. The search starts from the first phase and then goes through these phases sequentially. Each stage starts from the best solution found from the previous stage and tries to improve it, except the final stage RMIP which operates on a pool of elite solutions found in phase 2 and phase 3. In this application, the maximum size of the elite solution pool is fixed to 100. Once the pool reaches the maximum size, inferior solutions will be replaced with better solutions that are not in the elite pool yet. In the initialisation phase, if Cplex fails to generate a feasible solution, TA_MGLS will start from an initial solution generated by LP_Round, a procedure which solves the corresponding LP-relaxation of the problem and then rounds the design variables to the nearest integers. The flow variables are then recomputed by solving the corresponding CMMCF problem.

The underlining ideas for this hybridisation are: (1) for many problem instances, Cplex MIP solver can find good quality (in a few cases, optimal) solutions fairly quickly but often converges very slowly afterwards. In addition, Cplex MIP tends to fail to produce feasible solutions for instances with a large number of arcs. (2) The Tabu Assisted Multi-start Guided Local Search (TA_MGLS) (Bai et al., 2012) is very efficient in finding a good quality feasible solution thanks to the special ability of the guided local search which exploits the structure of the problem directly. (3) The κ-node neighbourhood has much better reachability than the previous neighbourhood. It can reach some local optima that otherwise cannot be found. (4) We observed, in our initial experiments, that the network design differences between the best neighbouring solutions found in stage two and stage three are not significant. Many of them share identical arc settings. In our hybrid algorithm, a reduced network design problem is modelled by prefixing the values of these identical arcs in the original MIP model. In our experiments, the reduced problem was generally solvable in 10–15 s for the majority of our tested instances with a few exceptions. We have set a time limit of 60 s for this stage (see Table 4) to ensure the majority of the computational time can be spent on the κ-node neighbourhood search stage

The hybrid algorithm is applied to solve the same instances. The time limits for the four stages are set to $t_1 = 2400, t_2 = 600, t_3 = 4140, t_4 = 60$ s, respectively. Therefore, the total CPU time for each run of the hybrid algorithm is 7200 s. The hybrid algorithm is run 5 times (each with a different random seed) on a PC with 2.0 GHz Intel Core 2 CPU and 8GB RAM and the best and the mean re-

Table 4
The results of the hybrid algorithm from the four stages, MIP_TL, TA_MGLS, TS_κ-node, and RMIP. Times permitted at these stages are $t_1 = 2400\ s, t_2 = 600\ s, t_3 = 4140\ s, t_4 = 60$ s, respectively. gap% is the relative gap to the best known solution.

Instance		MIP_TL	TA_MGLS		TS_κ_node		RMIP		Overall
			best	avg	best	avg	best	avg	best
c37	obj	98,829	98,829	98,829	97,737	98,163	97,274	97,767	97,274
	gap%	1.6%	1.6%	1.6%	0.5%	0.9%	0.0%	0.5%	
c38	obj	140,495	140,495	140,495	139,921	140,351	139,395	139,468	139,395
	gap%	0.8%	0.8%	0.8%	0.4%	0.7%	0.0%	0.1%	
c39	obj	100,478	100,478	100,478	100,478	100,478	100,478	100,478	100,478
	gap%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	
c40	obj	140,171	140,017	140,131	138,994	139,275	138,994	139,174	138,994
	gap%	0.8%	0.7%	0.8%	0.0%	0.2%	0.0%	0.1%	
c45	obj	78,054	78,037	78,049	77,674	77,826	77,463	77,658	77,463
	gap%	0.8%	0.7%	0.8%	0.3%	0.5%	0.0%	0.3%	
c46	obj	120,926	119,324	120,123	119,259	119,706	119,259	119,346	119,259
	gap%	1.4%	0.1%	0.7%	0.0%	0.4%	0.0%	0.1%	
c47	obj	76,208	76,208	76,208	76,208	76,208	76,208	76,208	76,208[a]
	gap%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	
c48	obj	112,449	112,449	112,449	111,475	112,000	111,475	111,475	111,475
	gap%	0.9%	0.9%	0.9%	0.0%	0.5%	0.0%	0.0%	
c49	obj	54,683	54,683	54,683	54,683	54,683	54,683	54,683	54,683[a]
	gap%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	
c50	obj	99,322	99,112	99,211	98,948	99,141	98,595	98,773	98,595
	gap%	0.7%	0.5%	0.6%	0.4%	0.6%	0.0%	0.2%	
c51	obj	53,030	53,030	53,030	53,030	53,030	53,030	53,030	53,030
	gap%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	
c52	obj	102,512	102,512	102,512	101,808	102,022	101,576	101,798	101,576
	gap%	0.9%	0.9%	0.9%	0.2%	0.4%	0.0%	0.2%	
c53	obj	115,452	115,384	115,384	115,330	115,330	114,891	114,962	114,891
	gap%	0.5%	0.4%	0.4%	0.4%	0.4%	0.0%	0.1%	
c54	obj	161,118	155,487	155,715	154,668	155,033	154,336	154,837	154,336
	gap%	4.4%	0.7%	0.9%	0.2%	0.5%	0.0%	0.3%	
c55	obj	118,441	117,891	117,975	117,295	117,699	117,141	117,527	117,141
	gap%	1.1%	0.6%	0.7%	0.1%	0.5%	0.0%	0.3%	
c56	obj	159,863	159,115	159,246	157,755	158,241	157,655	158,137	157,655
	gap%	1.4%	0.9%	1.0%	0.1%	0.4%	0.0%	0.3%	
c57	obj	48,693	48,693	48,693	48,693	48,693	48,693	48,693	48,693[a]
	gap%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	
c58	obj	61,732	61,647	61,710	61,494	61,601	61,433	61,434	61,433
	gap%	0.5%	0.3%	0.5%	0.1%	0.3%	0.0%	0.0%	
c59	obj	46,751	46,750	46,751	46,750	46,751	46,750	46,750	46,750
	gap%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	
c60	obj	56,269	56,269	56,269	56,252	56,252	56,207	56,241	56,207
	gap%	0.1%	0.1%	0.1%	0.1%	0.1%	0.0%	0.1%	
c61	obj	215,621	102,876	103,746	101,692	102,166	101,316	101,641	101,316
	gap%	112.8%	1.5%	2.4%	0.4%	0.8%	0.0%	0.3%	
c62	obj	405,452	148,627	150,385	145,571	146,812	145,185	146,447	145,185
	gap%	179.3%	2.4%	3.6%	0.3%	1.1%	0.0%	0.9%	
c63	obj	192,725	100,041	100,808	99,330	99,750	99,133	99,604	99,133
	gap%	94.4%	0.9%	1.7%	0.2%	0.6%	0.0%	0.5%	
c64	obj	137,015	135,873	136,407	134,720	135,437	134,122	134,916	134,122
	gap%	2.2%	1.3%	1.7%	0.4%	1.0%	0.0%	0.6%	
avg obj		120,679	100,993	101,220	100,407	100,694	100,221	100,460	100,221
avg gap%		16.9%	0.6%	0.8%	0.2%	0.4%	0.0%	0.2%	

[a] The optimal solution objective value.

sults are reported. Table 4 shows the detailed results of the four different stages of the hybrid algorithm and Table 5 compares the final results by the hybrid algorithm against those by Cplex12.4 MIP solver with 2 h time limit (Cplex_2h), and a very recent metaheuristic TS-PR (Minh et al., 2013), which was run on a workstation with AMD Dual-Core Opteron 2.4 GHz CPU and 16GB RAM. Due to data unavailability in the referenced article, only the best results out of 10 runs by TS-PR are included, each of which takes 7785 s CPU time on average. It can be seen that the proposed hybrid algorithm based on the κ-node neighbourhood performed competitively and has found new best solutions for several instances. It is particularly suitable as a quick post-optimisation approach for Cplex which appears to suffer slow convergence issues for some instances.

8. Conclusions and future work

Service network design is the core problem for freight transportation network planning and optimisation. The problem is strongly NP-Hard and is particularly challenging due to the complex constraints. Differing from the previous studies which have focused on more effective generic search strategies, this research proposed and studied a novel neighbourhood structure that permits simultaneous changes of multiple arcs incident upon a number of given nodes while maintaining the solution feasibility throughout the search. The new neighbourhood function, evaluated in the context of two basic metaheuristic approaches, showed better reachability than the existing arc-flipping neighbourhood functions.

Table 5
Results of the hybrid algorithm in comparison with Cplex and TS-PR. TS-PR (Minh et al., 2013) was run on a workstation with AMD Dual-Core Opteron 2.4 GHz CPU and 16GB RAM. The results are the best objective values out of 10 runs, with each run taking 7785 s CPU time on average. Both CPlex_2h and our hybrid algorithm was given 7200 s CPU time on a PC with 2.0 GHz Intel Core 2 CPU and 8GB RAM. Therefore, TS-PR consumes much more computational time than both CPlex and our hybrid algorithm. The hybrid algorithm was run five time and both the best, average and the worst results are reported here. The best performing algorithm for each instance are highlighted in **bold** and the best results are listed in the last column.

Instance		Cplex_2h		TS-PR		Hybrid Algorithm				best
id	feature	obj	gap%	best	gap%	best	gap%	avg	gap%	known
c37	C20,230,200,V,L	98,271	1.0%	**97,274**	0.0%	**97,274**	0.0%	**97,767**	0.5%	97,274
c38	C20,230,200,F,L	141,398	1.4%	**139,395**	0.0%	**139,395**	0.0%	**139,468**	0.1%	139,395
c39	C20,230,200,V,T	**100,221**	0.0%	100,720	0.5%	100,478	0.3%	100,478	0.3%	100,221[a]
c40	C20,230,200,F,T	139,278	0.2%	**138,962**	0.0%	138,994	0.0%	139,174	0.2%	138,962
c45	C20,300,200,V,L	77,907	0.6%	77,584	0.2%	**77,463**	0.0%	**77,658**	0.3%	77,463
c46	C20,300,200,F,L	120,926	1.4%	119,987	0.6%	**119,259**	0.0%	**119,346**	0.1%	119,259
c47	C20,300,200,V,T	**76,208**	0.0%	76,450	0.3%	**76,208**	0.0%	**76,208**	0.0%	76,208[a]
c48	C20,300,200,F,T	111,963	0.4%	111,776	0.3%	**111,475**	0.0%	**111,475**	0.0%	111,475
c49	C30,520,100,V,L	**54,683**	0.0%	54,783	0.2%	**54,683**	0.0%	**54,683**	0.0%	54,683[a]
c50	C30,520,100,F,L	99,101	0.5%	100,098	1.5%	**98,595**	0.0%	**98,773**	0.2%	98,595
c51	C30,520,100,V,T	**53,023**	0.0%	53,035	0.0%	53,030	0.0%	53,030	0.0%	53,023
c52	C30,520,100,F,T	101,599	0.2%	**101,412**	0.0%	101,576	0.2%	101,798	0.4%	101,412
c53	C30,520,400,V,L	114,983	0.1%	115,528	0.6%	**114,891**	0.0%	**114,962**	0.1%	114,891
c54	C30,520,400,F,L	154,295	0.6%	**153,409**	0.0%	154,336	0.6%	154,837	0.9%	153,409
c55	C30,520,400,V,T	**116,781**	0.0%	117,226	0.4%	117,141	0.3%	117,527	0.6%	116,781
c56	C30,520,400,F,T	158,307	1.5%	**155,906**	0.0%	157,655	1.1%	158,137	1.4%	155,906
c57	C30,700,100,V,L	**48,693**	0.0%	48,807	0.2%	**48,693**	0.0%	**48,693**	0.0%	48,693[a]
c58	C30,700,100,F,L	61,448	0.1%	**61,408**	0.0%	61,433	0.0%	61,434	0.0%	61,408
c59	C30,700,100,V,T	**46,750**	0.0%	46,812	0.1%	**46,750**	0.0%	**46,750**	0.0%	46,750
c60	C30,700,100,F,T	**56,177**	0.0%	56,237	0.1%	56,207	0.1%	56,241	0.1%	56,177
c61	C30,700,400,V,L	**99,493**	0.0%	100,583	1.1%	101,316	1.8%	101,641	2.2%	99,493
c62	C30,700,400,F,L	141,735	0.5%	**141,037**	0.0%	145,185	2.9%	146,447	3.8%	141,037
c63	C30,700,400,V,T	**97,748**	0.0%	97,875	0.1%	99,133	1.4%	99,604	1.9%	97,748
c64	C30,700,400,F,T	**133,387**	0.0%	133,686	0.2%	134,122	0.6%	134,916	1.1%	133,387
	overall avg	100,182	0.4%	100,000	0.3%	100,221	0.4%	100,460	0.6%	99,735

[a] denotes the optimal solution objective.

Due to the scale of the proposed neighbourhood size and the computational complexity of the solution evaluation, various techniques and heuristics have been designed to speed up the evaluation, including cut/set inequality conditions checking for candidate solutions with insufficient open arcs, approximate flow redistributing on a residual network, and partial solution evaluations.

Finally a hybrid algorithm based on the κ-node neighbourhood is developed and its results are compared against Cplex MIP solver and a recent metaheuristic method TS-PR. The results by the prosed hybrid algorithm are very competitive and some of them are the new best solutions. In future, we plan to extend the proposed new neighbourhood method to stochastic service network design problems which has similar constraints but much larger problem sizes and hence is more challenging to solve.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (Grant No. 71471092), Zhejiang Natural Science Foundation (Grant No. LR17G010001) and Ningbo Science and Technology Bureau (Grant No. 2011B81006, 2014A35006).

References

Ahuja, R., Magnanti, T., Orlin, J., 1993. Network Flows: Theory, Algorithms and Applications. Prentice Hall.
Alumur, S.A., Kara, B.Y., Karasan, O.E., 2012. Multimodal hub location and hub network design. Omega 40 (6), 927–939. doi:10.1016/j.omega.2012.02.005.
Andersen, J., Christiansen, M., Crainic, T.G., Gronhaug, R., 2011. Branch and price for service network design with asset management constraints. Transp. Sci. 45 (1), 33–49.
Andersen, J., Crainic, T.G., Christiansen, M., 2009. Service network design with management and coordination of multiple fleets. Eur. J. Oper. Res. 193 (2), 377–389.
Armacost, A.P., Barnhart, C., Ware, K.A., 2002. Composite variable formulations for express shipment service network design. Transp. Sci. 36 (1), 1–20.
Bai, R., Kendall, G., Qu, R., Atkin, J., 2012. Tabu assisted guided local search approaches for freight service network design. Inf. Sci. 189, 266–281.
Bai, R., Wallace, S., Li, J., Chong, A., 2014. Stochastic service network design with rerouting. Transp. Res. Part B 60, 50–65.
Bai, R., Xue, N., Chen, J., Roberts, G.W., 2015. A set-covering model for a bidirectional multi-shift full truckload vehicle routing problem. Transp. Res. Part B 79, 134–148.
Barcos, L., Rodriguez, V., Alvarez, M., Robuste, F., 2010. Routing design for less-than-truckload motor carriers using ant colony optimization. Transp. Res. Part E 46, 367–383.
Barnhart, C., Krishnan, N., Kim, D., Ware, K., 2002. Network design for express shipment delivery. Comput. Optim. Appl. 21, 239–262.
Carle, M.A., Martel, A., Zufferey, N., 2012. The CAT metaheuristic for the solution of multi-period activity-based supply chain network design problems. Int. J. Prod. Econ. 139 (2), 664–677. doi:10.1016/j.ijpe.2012.06.016.
Chouman, M., Crainic, T.G., 2014. Cutting-plane meta-heuristic for service network design with design-balanced requirements. Transp. Sci. 49 (1), 99–113.
Crainic, T.G., 2000. Service network design in freight transportation. Eur. J. Oper. Res. 122 (2), 272–288.
Crainic, T.G., Gendreau, M., Farvolden, J.M., 2000. A simplex-based tabu search method for capacitated network design. INFORMS J. Comput. 12 (3), 223–236.
Crainic, T.G., Gendreau, M., Soriano, P., Toulouse, M., 1993. A tabu search procedure for multicommodity location/allocation with balancing requirements. Ann. Oper. Res. 41 (1–4), 359–383.
Crainic, T.G., Kim, K.H., 2007. Chapter 8 intermodal transportation. In: Barnhart, C., Laporte, G. (Eds.), Transportation, Handbooks in Operations Research and Management Science, 14. Elsevier, pp. 467–537.
Crainic, T.G., Rousseau, J.M., 1986. Multicommodity, multimode freight transportation: a general modeling and algorithmic framework for the service network design problem. Transp. Res. Part B 20 (3), 225–242.
Crainic, T.G., Roy, J., 1988. Or tools for tactical freight transportation planning. Eur. J. Oper. Res. 33 (3), 290–297.
Ghamlouche, I., Crainic, T.G., Gendreau, M., 2003. Cycle-based neighbourhoods for fixed-charge capacitated multicommodity network design. Oper. Res. 51 (4), 655–667.
Ghamlouche, I., Crainic, T.G., Gendreau, M., 2004. Path relinking, cycle-based neighbourhoods and capacitated multicommodity network design. Ann. Oper. Res. 131, 109–133.
Glover, F., Laguna, M., 1997. Tabu Search. Kluwer academic publishers, Boston.
Hoff, A., Lium, A.G., Lø kketangen, A., Crainic, T., 2010. A metaheuristic for stochastic service network design. J. Heuristics 16 (5), 653–679.

Kendall, G., Bai, R., Blazewicz, J., De Causmaecker, P., Gendreau, M., John, R., Barry McCollum, J., Pesch, E., Qu, R., Sabar, N., Vanden Berghe, G., Yee, A., 2016. Good laboratory practice for optimization research. J. Oper. Res. Soc. 67 (4), 676–689.

Kleeman, M.P., Seibert, B.A., Lamont, G.B., Hopkinson, K.M., Graham, S.R., 2012. Solving multicommodity capacitated network design problems using multi-objective evolutionary algorithms. IEEE Trans. Evol. Comput. 16 (4), 449–471. doi:10.1109/TEVC.2011.2125968.

Minh, V.D., Crainic, T.G., Toulouse, M., 2013. A three-stage metaheuristic for the capacitated multi-commodity fixed-cost network design with design-balance constraints. J. Heuristics 19 (5), 757–795.

Nickel, S., Saldanha-da Gama, F., Ziegler, H.P., 2012. A multi-stage stochastic supply network design problem with financial decisions and risk management. Omega 40 (5), 511–524.

Pedersen, M.B., Crainic, T.G., Madsen, O.B., 2009. Models and tabu search metaheuristics for service network design with asset-balance requirements. Transp. Sci. 43 (2), 158–177.

Powell, W.B., 1986. A local improvement heuristic for the design of less-than-truckload motor carrier networks. Transp. Sci. 20 (4), 246–257.

Voudouris, C., Tsang, E.P., 2003. Guided local search. In: Glover, F., Kochenberger, G. (Eds.), Handbook of Metaheuristics. Kluwer, pp. 185–218.

Wieberneit, N., 2008. Service network design for freight transportation: a review. OR Spectr. 30, 77–112.

Yaghini, M., Momeni, M., Sarmadi, M., 2012. A simplex-based simulated annealing algorithm for node-arc capacitated multicommodity network design. Appl. Soft Comput. 12 (9), 2997–3003. doi:10.1016/j.asoc.2012.04.022.