INFOR: INFORMATION SYSTEMS AND OPERATIONAL RESEARCH, 2017
VOL. 55, NO. 1, 16–37
http://dx.doi.org/10.1080/03155986.2016.1262580

Taylor & Francis
Taylor & Francis Group

A scheme for determining vehicle routes based on Arc-based service network design

Xiaoping Jiang [a], Ruibin Bai[a], Jason Atkin [b] and Graham Kendall [b,c]

[a]School of Computer Science, University of Nottingham Ningbo China, Ningbo, China; [b]School of Computer Science, University of Nottingham, Nottingham, UK; [c]School of Computer Science, University of Nottingham Malaysia Campus, Semenyih, Malaysia

ABSTRACT

In freight transportation, less-than-truckload carriers often need to assign each vehicle a cyclic route so that drivers can come back home after a certain period of time. However, the Node-Arc model for service network design addresses decisions on each arc and does not determine routes directly, although the vehicle balancing constraint ensures that the number of outgoing vehicles equals the number of incoming vehicles at each node. How to transform the optimized service network into a set of vehicle routes remains an important problem that has not yet been studied. In this paper, we propose a three-phase scheme to address this problem. In the first stage, we present an algorithm based on the depth-first search to find all of the different cyclic routes in a service network design solution. In the second stage, we propose to prune poor cyclic routes using real-life constraints so that a collection of acceptable vehicle routes can be obtained before route assignment. Some of the pruning can also be done in the first stage to speed up the proposed algorithm. In the third stage, we formulate the problem of selecting a set of cyclic routes to cover the entire network as a weighted set covering problem. The resulting model is formulated as an integer program and solved with IBM ILOG CPLEX solver. Experimental results on benchmark instances for service network design indicate the effectiveness of the proposed scheme which gives high-quality solutions in an efficient way.

ARTICLE HISTORY
Received 3 February 2016
Accepted 1 September 2016

KEYWORDS
Service network design; depth-first search; pruning; set covering; integer programming

1. Introduction

Freight transportation is fundamental to economic prosperity and daily life in modern society. Over the past few years, less-than-truckload (LTL) transportation and express delivery, for which shipments typically weigh no more than 10,000 lbs, have seen significant increase in freight flows in the wake of online shopping (Bai et al. 2014). In order to succeed in this highly competitive market, LTL carriers have to offer consumers high-quality services at a lower price. Therefore, it is imperative for carriers to set up service plans so as to optimize the utilization of critical logistic assets (vehicles, warehouses, etc.) and reduce the overall cost.

CONTACT Ruibin Bai ruibin.bai@nottingham.edu.cn

Freight transportation planning is generally carried out at three levels: strategic, tactical and operational (Crainic 2000). At the tactical planning level, LTL carriers need to make decisions about a set of paths via which each commodity flows from its origin to the destination. A path is a sequence of nodes which represent geographically distributed end-of-line and break-bulk terminals. In addition, LTL carriers usually choose to consolidate shipments from different customers into truckloads of freight at various terminals in order to spread transportation costs over as many customers as possible. Network design could be introduced to assist the tactical planning, which is also called the service network design problem (SNDP) (Wieberneit 2008).

In the SNDP, we consider an underlying network of geographically distributed end-of-line and break-bulk terminals (often cities, or regional transportation hubs). Break-bulk terminals are those where freight is unloaded, consolidated and then reloaded onto the same or different vehicles. Some of these terminals are connected by links (e.g. truck lines), over which freight flows. To meet customers' demands, a carrier hauls multiple commodities through the network from many origin terminals to many destination terminals. The services are performed by a number of vehicles, each with a limited capacity. As each link is associated with a fixed cost and a variable cost, the carrier's objective is to deliver these goods at minimum possible cost. Thus, it needs to make decisions about which links to open and how shipments should be routed through the network. The selected links together with associated terminals comprise a service network, which is expected to satisfy customers' demands at minimum total cost. The fixed cost represents the minimum level of resources needed to open a link, like capital investment for purchasing vehicles and employing staff, while the variable cost such as fuel consumption and tolls is incurred per unit by hauling commodities.

The most common model for service network design is the Node-Arc model, of which more details are given in Section 3. It has two types of decision variables, namely design variables and flow variables. Design variables represent the number of logistic assets (e.g. the frequency of vehicles for carrying the load) on a link between two nodes. The link is also referred to as an arc. Flow variables indicate the amount of commodity flow on an arc. In other words, the Node-Arc model defines decision variables for each arc. As a result, the service network design solution addresses decisions on arcs. Figure 1 gives an example of an optimized service network from the Node-Arc formulation. Arcs represented by solid lines are selected to comprise the optimized network and closed arcs are represented by

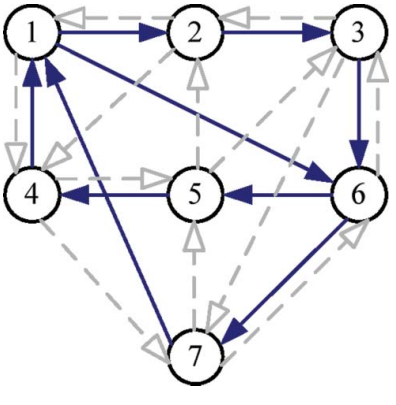

Figure 1. A simple example of an optimized service network.

dashed lines. Note that although we mainly deal with the general Node-Arc model in the following parts of this paper, algorithms developed for the general Node-Arc model can be adapted to the time-space SNDP models. The rationale behind this is that a time-space SNDP model (Pederson et al. 2008) can be transformed into a general Node-Arc model.

However, it remains difficult for LTL carriers to directly adopt the optimized service network from the Node-Arc model. In practice, LTL carriers use a large number of vehicles to operate services and need to assign each vehicle a route. A vehicle route is typically a sequence of network nodes which includes several arcs rather than only one arc. A route should be feasible for a given vehicle to travel for the entire journey, from the first to the last node. Due to various restrictions, routes must often take the form of cycles so that drivers can come back home after a certain period of time. Although the vehicle balancing constraint ensures that the number of departing vehicles equals the number of incoming vehicles (Vu et al. 2013), the Node-Arc service network design model does not directly determine routes for vehicles. This is very different from the classic vehicle routing problem (VRP). The SNDP permits freight consolidation and transfers at intermediate nodes and hence distinguishes between a vehicle route (the list of nodes visited by a vehicle) and a commodity path (the list of nodes that a commodity flows through). A vehicle route can be used to service multiple commodities and a commodity can be delivered through a combination of multiple vehicles with options of transfers at intermediate nodes. In the classic VRP, a commodity is normally serviced by a single vehicle. Furthermore, a variety of other real-life constraints may arise when putting transport operations into certain contexts, such as working time regulations, fairness between drivers, and so on. From the perspective of users, it is thus of primary importance to be able to transform a service network design solution based on the Node-Arc formulation into practical routes.

This paper aims to address this research gap. The main contribution is twofold. First, it introduces a new problem of how to generate a set of vehicle routes based on an optimized service network. It attempts to transform the tactical planning results into vehicle routes, which is valuable for operational planning. It is expected to promote further research on this important problem. Second, it provides an efficient decomposition strategy with three phases to determine vehicle routes. One huge benefit of decomposition is that it enables pruning poor cyclic routes prematurely so that some real-life constraints can be satisfied before route assignment.

The remainder of this paper is organized as follows. In Section 2, we review related literature on service network design and crew scheduling. A detailed description of our research question is given in Section 3. In Section 4, we present an algorithm to search for all of the different cyclic routes in a solution network. In Section 5, we propose to prune poor cyclic routes before route assignment. In Section 6, we develop a method to cover a solution network with the cyclic routes. Experimental results are reported in Section 7. In Section 8, we give our concluding remarks. The potential limitations of our research and some directions for further study are also presented.

2. Literature review

2.1. Service network design

Service network design has been widely used to address tactical freight flow planning for LTL carriers. There are two prominent network design formulations in the scientific

literature, the Node-Arc model and the Path-based model (Crainic 2000). The Path-based model, in which flow variables are defined on each path instead of on each arc, by its nature, is equivalent to the Node-Arc model where both design variables and flow variables are defined in terms of arcs. Both of the formulations can be modelled as mixed integer programs. It is not practical to solve either directly using integer programming methodologies for real-life problems on a medium or large scale (Bai et al. 2010) due to the computational resources required. Thus, research has focused on methodologies to obtain good quality solutions. Various techniques have been investigated and proposed, including decomposition (Costa 2005), column generation (Smilowitz et al. 2003) and meta-heuristics (Ghamlouche et al. 2004; Crainic & Li 2006). More comprehensive reviews can be found in Wieberneit (2008). Aside from deterministic models, some research has been done on stochastic service network design (Thapalia 2011; Bai et al. 2014).

Recent studies on service network design have tended to incorporate operational issues into the two classic models. Vehicles are taken into consideration explicitly when designing service networks and a new layer called asset management is integrated with the traditional design layer and flow layer. The asset studied in most of the literature is limited to the vehicles that are needed for the operation of the transportation service, although it may also include crew assignments. Pedersen et al. (2008) extended the traditional service network design models by introducing a design balance constraint. The constraint requires that the number of arcs entering a node must be equal to that leaving a node, which can be interpreted there being an equal number of vehicles entering and leaving each terminal. Motivated by the need to change vehicles at borders for intermodal transport networks, Andersen et al. (2009) proposed a more comprehensive service network design model. The proposed model introduced synchronization of multiple fleets that cover services. Crainic et al. (2016) enlarged the range of asset management aspects included in the traditional models and explicitly accounted for the limited number of resources available at each terminal. The common highlight of these papers is considering vehicles explicitly in the service network models. Although several methods have been proposed to solve these models (Teypaz et al. 2010; Andersen et al. 2011), this remains difficult as these are NP-hard problems. This paper is also motivated by considering vehicles explicitly in the Node-Arc model, but deals with it in a different way.

2.2. Crew scheduling

Another research area related to this paper is crew scheduling and rostering in the management of large transit systems (Erera et al. 2008), where a given set of trips has to be covered by a set of pairings and the corresponding overall cost is minimized. A pairing is a sequence of trips that can be conducted by a single crew (Caprara et al. 1999). The crew scheduling problem is formulated as a set covering problem in a significant body of research (Azadeha et al. 2013; Bai et al. 2015). A similar set covering problem is introduced into this paper to determine a set of cyclic routes that cover each arc in the Node-Arc model solution. As an NP-hard combinatorial optimization problem, the set covering problem has been extensively studied (Groiez et al. 2014). The proposed solutions can be divided into two classes: exact algorithms and heuristic algorithms (Sundar & Singh 2012). Exact algorithms aim to find the optimal solution, while heuristic algorithms aim to find a good or near-optimal solution in a reasonable time. Problems which are typically encountered in the real world are generally too large to be solved exactly in an acceptable

computation time. For this reason, heuristic algorithms such as greedy algorithms, genetic algorithms, simulated annealing, ant colony optimization, particle swarm optimization and artificial bee colony (Sundar & Singh 2012) have been the focus of more and more research. For all of their simplicity, greedy algorithms do not in general produce sufficiently good solutions due to their myopic nature. Although other (non-greedy) heuristic algorithms may yield better results, they would often require unacceptable computing time for a typical LTL carrier with a few thousand drivers and 10,000 dispatch tasks (Erera et al. 2008). These algorithms usually involve complicated procedures, such as fine tuning of parameters. The set covering problem can be modelled as an integer program. As a state-of-the-art solver for integer programming, IBM ILOG CPLEX, provides an easier yet efficient avenue to solve the weighted set covering problem of small and medium sizes. In this paper, CPLEX is utilized as the main tool for the set covering problem. Of course, for practitioners who want to adopt the route generation scheme in this paper, this part can be replaced by other efficient algorithms for the set covering problem. The objective of this paper is not to propose a new algorithm for the set covering problem.

3. Problem description

3.1. Node-Arc model

As mentioned earlier, the most common model for service network design is the Node-Arc model (Crainic 2000). Consider a directed graph $G=(N,A)$ where N is the node set and A is the arc set. Let k be a commodity in the set $k \in K$. Let $d^k(d^k>0)$ denote the quantity of k, while $O_k \in N$ and $D_k \in N$ represent its origin and destination, respectively. Let b_i^k be the net outflow which indicates the net quantity of commodity k flowing outwards at node i. Let u^f denote the capacity of vehicle type $f \in F$ where F is the set of all transport vehicle types. h_{ij}^f is the fixed cost for each use of vehicle type f on arc $(i,j) \in A$, whereas c_{ij}^k represents the variable cost of shipping a unit of commodity k along arc (i,j). There are two decision variables, denoted by x_{ij}^k and y_{ij}^f, respectively. Continuous variables x_{ij}^k represent the flow of commodity k on arc $(i,j) \in A$. Design variables y_{ij}^f represent the frequency of vehicle type f on arc (i,j) and are discrete. The Node-Arc model can be modelled as follows (Bai et al. 2010):

$$\min \sum_{f\in F}\sum_{(i,j)\in A} h_{ij}^f y_{ij}^f + \sum_{k\in K}\sum_{(i,j)\in A} c_{ij}^k x_{ij}^k \tag{1}$$

s.t.

$$\sum_{k\in K} x_{ij}^k \le \sum_{f\in F} u^f y_{ij}^f, \forall (i,j) \in A \tag{2}$$

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

$$\sum_{j\in N} y_{ij}^f - \sum_{j\in N} y_{ji}^f = 0, \forall i \in N, \forall f \in F \tag{4}$$

$$x_{ij}^k \ge 0,\ \forall k \in K, \forall (i,j) \in A \tag{5}$$

$$y_{ij}^f \ge 0 \text{ and integer}, \forall f \in F, \forall (i,j) \in A \tag{6}$$

The net outflow b_i^k is defined as follows:

$$b_i^k = \begin{cases} d^k & \text{if } i = O_k, \\ -d^k & \text{if } i = D_k, \ \forall i \in N, \forall k \in K \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

The objective is to minimize the sum of fixed costs and variable costs induced by shipping all commodities from their origins to destinations. Constraint (2) ensures that the total flow along each arc does not exceed its capacity. Constraint (3) is the flow balance constraint, making sure that all commodities are shipped from their origins to destinations. Constraint (4) is the design balance constraint (Pederson et al. 2008; Vu et al. 2013), which ensures all vehicles finally return to their origins to get ready for next service period. Constraints (5) and (6) are used to guarantee the non-negativity of decision variables and the integrity of design variables.

3.2. Determining vehicle routes

Once the two decision variables of the Node-Arc model have been assigned values, an optimized service network can be obtained. A very small example involving seven nodes is shown in Figure 1, although there are typically a large number of nodes for a LTL carrier. At each node, there are equal numbers of inbound and outbound arcs.

The design balance constraint actually models a common fact that a logistic asset is associated with a specific terminal in a transportation network and must return there before the next service period. For instance, drivers in LTL businesses must periodically return to the terminal closest to their homes according to federal or labour union regulations (Crainic et al. 2016). As a result, this constraint induces cyclic structures in the service network, which means that the numbers of inbound and outbound arcs are equal for each node.

In most practical cases, LTL carriers need to assign each vehicle a route to operate services. Due to various restrictions, routes are often in the form of cycles so that drivers can come back home after a certain period of time. On the other hand, the cyclic routes need to be assigned in a way that fulfils the demands from customers. Because the flow balance constraint in the Node-Arc model ensures that all the commodities are shipped from their origin to their destination, the service network design solution is certain to meet all customer demands. As a consequence, every arc in the solution network should be covered by any final route assignment, otherwise, the assigned routes will not be sufficient to satisfy all of the demands. In summary, to assign each vehicle a cyclic route and keep services running well for carriers, we need to transform the optimized service network into a set of cyclic routes that cover all the arcs.

In this paper, we decompose the problem into three phases. In the first stage, we aim to find all of the different cyclic routes in an optimized service network. For example, in Figure 1, there are four cyclic routes, which are shown in Figure 2. To ascertain this, we first need to represent and store the optimized service network,

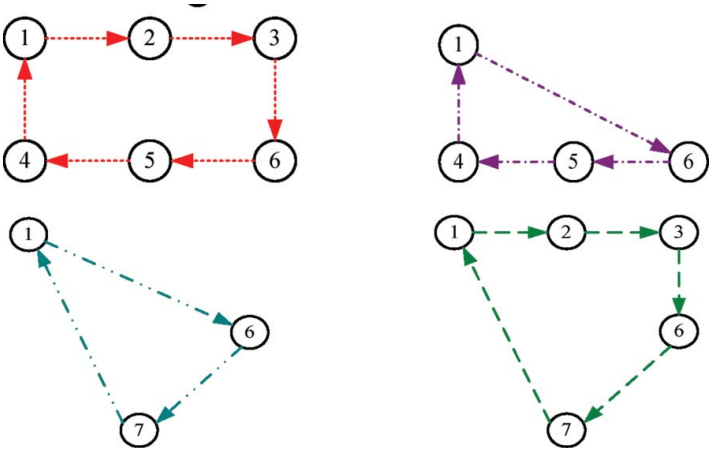

Figure 2. Cyclic routes included in Figure 1.

which is a directed graph. We also need to propose an algorithm that can find all of the different cyclic routes in a given directed graph. Due to some real-life constraints, some of the cyclic routes may be unacceptable for practical applications. In view of this, we propose to prune poor cyclic routes in the second stage before route assignment. In the third stage, we aim to find a set of cyclic routes that cover each arc in the network. Usually there will be various ways to cover the network and we want to find the most cost-effective one for LTL carriers. For instance, Figure 3 indicates two ways to cover the network in Figure 1, using either the left two cyclic routes or the right two in Figure 2. Therefore, we also need to design metrics to measure the cost of each route, such as the total distance, the number of nodes and so on. And then we need to find a set of cyclic routes that cover the entire network while minimizing the total cost.

Although the standard Node-Arc model does not take into account time-related constraints explicitly, these can be handled implicitly through a time-space network (Pederson et al. 2008), which can be linearly transformed into a standard Node-Arc formulation. Therefore, our scheme is applicable for both scenarios (i.e. with and without time related constraints).

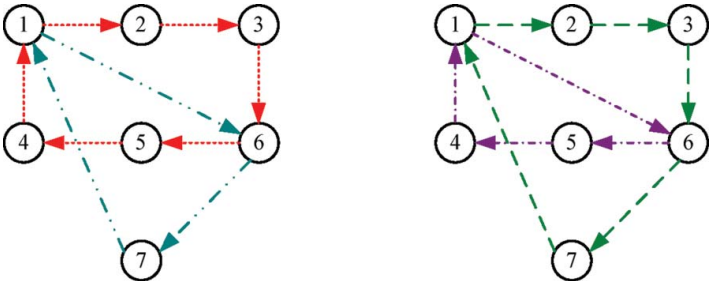

Figure 3. Two different ways to cover Figure 1 using cyclic routes in Figure 2.

3.3. SNDP versus VRP

When discussing vehicle routes, it is natural to think about the well-known VRP (Barcos et al. 2010; Benslimane & Benadada 2013). To bring a clearer picture of the SNDP, it is worth making a comparison between these problems. The SNDP differs from the VRP in several aspects, which are discussed below.

First and foremost, the presence of break-bulk terminals, which perform the consolidation or transshipment function (Crainic 2000), makes the SNDP distinctly different. In the SNDP, commodities shipped by different vehicles are gathered at the break-bulk terminal to be sorted, grouped and reloaded. From the point of view of commodities, break-bulk terminals are neither origins nor destinations, but rather those nodes through which commodities flow before reaching the destination terminals. By contrast, the nodes in most VRPs are either origins or destinations, and sometimes both in some of its variants, such as the pickup and delivery problem with a many-to-many structure (Berbeglia et al. 2007; Cordeau et al. 2008). Each node is visited exactly once by exactly one vehicle (Laporte 1992). Thus, the SNDP is more suitable for LTL transport because commodity transshipment and consolidation can be readily modelled. However, the SNDP has its own downsides, one of which is a lack of explicit representation of the vehicle route constraints, especially for the Node-Arc SNDP model. This disadvantage motivates our work of determining vehicle routes for LTL carriers based on the solution obtained from the Arc-based SNDP model.

Second, the VRP is in general based on a complete graph (or digraph). In this graph, every pair of distinct nodes is connected by a link over which freight flows. However, this assumption does not hold for the underlying network in the SNDP. Typically, only some of the terminals are connected to each other therein.

Third, the traditional VRP has a constant cost on each arc, which is normally associated with the travel distance or time. Although each arc also has a corresponding cost in the SNDP, the cost varies with the amount of the commodities flowing on the arc. Accordingly, the decision variables of the SNDP involve the flow of a commodity on an arc, whereas those of the traditional VRP do not. The rationale behind this can be explained as follows: in the SNDP, due to the consolidation process, what a vehicle carries may change at the break-bulk terminal. Without this decision variable, variations in the types and amounts of the commodity it hauls are unknown. Even if the arc selection is finalized, there will be a wide range of possibilities to consider in terms of the commodity and we still cannot determine the most cost-effective commodity flow. In the traditional VRP, as each node has a clear demand, a finalized arc selection will bring out a definite commodity flow.

Finally, the majority of VRP variants are characterized by having one or more depots where vehicle routes start and end (Laporte 1992; Cordeau & Laporte 2003). Unlike the VRP, the general SNDP is not under such a constraint. Even in the SNDP with a design balance constraint, which induces cyclic structures, the resulting cycles usually do not have a common depot. One may ask, if this is the case, then where are the vehicles based? This question actually falls within the area of empty balancing (Crainic 2000), another significant problem for the SNDP. However, if this problem is not the centre of attention, it is typically assumed that vehicles are well repositioned beforehand so that there are sufficient supplies of vehicles at terminals of the underlying network. The problem considered here makes this assumption.

4. Finding all the different cyclic routes

4.1. Representation of networks

As mentioned in Section 3, in order to find all of the different cyclic routes, we first need to represent and store the optimized service network. An optimized service network is a directed graph (Ahuja et al. 1993), enabling us to utilize ideas from graph theory.

Either an adjacency matrix or adjacency list could be adopted to represent and store a directed graph (Ahuja et al. 1993). Although an adjacency matrix has the merit of simplicity, the space requirement can be prohibitive (Weiss 2013) if the graph is large and sparse, which is often the case with the domain under investigation. A better representation for a sparse graph is an adjacency list, in which each node stores a list of all of its adjacent nodes. Given a limited number of exiting arcs, as is usually the case in practice, the space requirement is linear in the size of the graph. In addition, an adjacency list makes it possible to get the list of adjacent vertices for a given node within constant time. Regarding the implementation of adjacency list in C++, we could choose either linked lists or arrays since it makes little difference to the performance of our scheme. Note that in Section 4.2, we use linked list to explain the proposed algorithm.

4.2. Searching for all the different cyclic routes

In order to find all of the different cyclic routes in a certain solution network, we propose to first find all of the cyclic routes starting from a given node. Note that a cyclic route in the first stage of finding all of the different cyclic routes does not have any repetition of nodes, which is referred to as a directed cycle (Ahuja et al. 1993) in graph theory, because we want to ensure that our scheme is also applicable for time-space-based service networks in which each node is a copy of a physical node at a particular time point. Any directed walk (Ahuja et al. 1993) whose last node is the same as its starting node can be obtained by a combination of cyclic routes. For example, the directed walk $1\rightarrow6\rightarrow7\rightarrow1\rightarrow6\rightarrow5\rightarrow4(\rightarrow1)$ in Figure 1 is not seen as a cyclic route because nodes 1 and 6 have been visited more than once. In fact, it is the combination of two cyclic routes, $1\rightarrow6\rightarrow7(\rightarrow1)$ and $1\rightarrow6\rightarrow5\rightarrow4(\rightarrow1)$. For convenience, we do not distinguish between such directed walks and cyclic routes in the second stage of pruning and the third stage of covering a network. These are all referred to as cyclic routes, but allow repeated nodes.

To detect a cyclic routes starting from a given node in a directed graph, we need to perform a traversal over the entire graph. As an effective graph traversal algorithm, depth-first search (Tarjan 1972) is adopted. In the context of searching cyclic routes, depth-first search will proceed from a particular root node as far as it can until this root node or a dead end is reached. Then depth-first search will backtrack to the last node from which a different path could have been taken. The search continues with this node until it has to backtrack again. Continuing in this fashion, all of the cyclic routes starting from this root node will be detected eventually.

In the following step, conceptually we could simply apply the same method to all of the other nodes to obtain all of the cyclic routes in this network. However, such a procedure would produce a large number of repeated cyclic routes which contains the same nodes placed in the same relative order. The only differences among these routes are their starting nodes. In Figure 1, such routes as $1\rightarrow6\rightarrow7(\rightarrow1)$ and $6\rightarrow7\rightarrow1(\rightarrow6)$ occur multiple

times (three in this case). They are essentially the same and hence we need to remove redundancy. To achieve this, normalization of all the routes is needed to facilitate the detection of redundancy. For example, we could rearrange the nodes of a cyclic route so that the node ranking the lowest is set as the starting one. Thus, routes like $1\rightarrow 6\rightarrow 7$ and $6\rightarrow 7\rightarrow 1$ are normalized as a common route $1\rightarrow 6\rightarrow 7$. There is no doubt the whole process involves huge amounts of repetitive work. Such an intuitively plausible method would lead to prohibitive consumption of memory and computational time in practice. Inspired by the normalization process above, we prove a proposition.

Proposition 4.1: *Suppose a set S which comprises all of the cyclic routes starting from a node v has already been found. When searching for cyclic routes starting with a node different from v, if v is included in a newly found cyclic router, then* $r \in S$.

Proof: Assume that we are searching for cyclic routes starting from a node w_1, which is different from v. A newly found cyclic route r including node v can be represented as

$$w_1 \rightarrow w_2 \rightarrow \cdots \rightarrow w_h \rightarrow v \rightarrow w_{h+1} \rightarrow \cdots (\rightarrow w_1) \tag{8}$$

We can rearrange these nodes in Equation (8) and get a cyclic route named r_{equi}, which starts with v, as shown in Equation (9).

$$v \rightarrow w_{h+1} \rightarrow \cdots \rightarrow w_1 \rightarrow w_2 \rightarrow \cdots \rightarrow w_h (\rightarrow v) \tag{9}$$

As the set S comprises all of the cyclic routes starting from v, we have $r_{\text{equi}} \in S$. Routes r and r_{equi} have the same nodes placed in the same relative order, so they are equivalent and represent the same cyclic route. Thus, we have $r = r_{\text{equi}}$. Combining $r_{\text{equi}} \in S$ and $r = r_{\text{equi}}$, we obtain the result that $r \in S$. This completes the proof of the proposition.□

Based on Proposition 4.1, a better method is proposed. When searching for cyclic routes starting from a node, depth-first search will check whether the node being reached has already been used as a starting node before. If so, depth-first search will bypass the node and backtrack earlier than normal. If not, the search procedure will keep going normally. Adding the evaluation of this condition into depth-first search ensures that no cyclic route would be searched twice. Searching time is thus reduced significantly. After the method of finding all the cyclic routes with a certain starting node is applied to all the nodes, all the cyclic routes generated are unique and the final set contains no duplicate route. Therefore, the cumbersome and time-consuming process of removing redundancy can be avoided.

In respect of implementation, we need to keep a record of all nodes that have been used as starting nodes for the purpose of evaluating the condition proposed above. An equivalent yet easier alternative is to number the nodes. We then search for cyclic routes in the order that the node with a lower index will be used as a starting node earlier. When exploring a cyclic route, if we reach a node with a lower index than the current starting node then abort the route immediately. Regarding the implementation of depth-first search, a recursive function was adopted. The most important part of doing depth-first search is to ensure that the algorithm does not run *ad infinitum*. To achieve that in C++, we keep track of the nodes which have already been visited by marking them.

Algorithm 1: Searching for all of different cyclic routes

input: a service network T, the node set V, an adjacent node *adjnode* of a given node v, a Boolean array *visited*.

Begin

1: number all of the nodes in V so that each has an index;
2: **for** all nodes $v \in V$ in ascending order of index **do**
//mark all nodes as unvisited
3: **memset**(*visited* , False, **sizeof**(*visited*));
//mark current starting node as visited
4: $visited[v] = \text{True}$;
//call predefined recursive function by value
5: $FindDiffCyclesFromANode(T, v)$;
6: **end for** all nodes
//declare function prototype
7: *void FindDiffCyclesFromANode*(network T, int v)
8: { define a pointer p pointing to an adjacent node of v;
9: **while** p is not null **do**
10: **if** $(p \rightarrow adjnode)$ is current starting node **then**
11: a cyclic route is found;
12: **else if** $(visited[p \rightarrow adjnode])$ is False **then**
13: **if** $(p \rightarrow adjnode) >$ (index of v) **then**
14: { $visited[p \rightarrow adjnode]$ =True;
//recursively call function
15: $FindDiffCyclesFromANode(T, p \rightarrow adjnode)$;
16: $visited[p \rightarrow adjnode]$ = False;}
17: make p point to next adjacent node of v;
18: **end while**}

end

Figure 4. Pseudo code for searching for all of different cyclic routes.

The pseudo code for searching for all of different cyclic routes is shown in Figure 4. Assume that we have an optimized service network T. It is represented by an adjacency list and implemented with linked lists. V is the node set of this network. A Boolean array *visited* is used to keep track of whether each node has been visited or not. *FindDiffCyclesFromANode*(T, v) is a function defined to search for all of the different cyclic routes starting from a node v. A newly found cyclic route is different from any previous one. Lines starting with '//' are comments.

5. Pruning of poor cyclic routes

When allocating vehicle routes, LTL carriers are faced with many real-life restrictions, some of which must not be violated. For instance, working time regulations for drivers in the European Union require that the daily driving time between the end of one daily rest period and the beginning of the following daily rest period should not exceed 9 hours when a vehicle is manned by one driver (Goel & Gruhn 2006). LTL carriers are monitored and a violation of the legislation would be severely fined (Rancourt et al. 2012). Similar

examples include delivery services with time windows (Kok et al. 2010; Spoorendonk & Desaulniers 2010). The total driving time during a journey is to a great extent determined by the length of a cyclic route. Therefore, the presence of such hard constraints is very likely to make some of the cyclic routes generated in Section 4 unacceptable in practice. Motivated by this, we propose to prune poor cyclic routes before determining vehicle routes to cover the entire network. In this way, we can possibly eliminate some issues for practical applications before route assignment and select from a collection of acceptable cyclic routes for the next stage.

Applying hard constraints to prune poor cyclic routes can be done either during the generation of cyclic routes in the first stage or as an intermediate stage after the first stage. Take drivers' working hours as an example. In the first stage, we can abort the search for a certain cyclic route prematurely when it already contains too many arcs to satisfy the prescribed constraint of daily driving time. This pruning can be achieved by adding a condition evaluating process after Step 13 in Algorithm 1. The condition is whether or not the accumulated driving hours along the arcs already included in the vehicle route exceed the time limit. The pruning strategy will help reduce the total search time and speed up Algorithm 1 in Figure 4.

Pruning as an intermediate stage after the first stage would have the advantage of knowing what other cycles have been generated already before pruning. Suppose we have found two cyclic routes, namely $A\rightarrow B\rightarrow C\rightarrow G(\rightarrow A)$ and $C\rightarrow D\rightarrow E\rightarrow F(\rightarrow C)$, in an optimized service network, as shown in Figure 5. They have a node 'C' in common and the whole figure is like '8' in shape. In the case where either route is relatively short, it would be more efficient for LTL carriers to assign both cyclic routes to the same vehicle compared with using two vehicles. Thus, we can combine these two cyclic routes to make a big and efficient one, that is, $A\rightarrow B\rightarrow C\rightarrow D\rightarrow E\rightarrow F\rightarrow C\rightarrow G(\rightarrow A)$. A combination of poor cyclic routes will be used as a whole for the third stage instead of separate ones. This approach can be seen as another way of pruning in a broad sense because it reduces the number of cyclic routes for the third stage.

A good pruning depends on the precise details of relevant restrictions imposed on a certain LTL carrier. Considering this, we intend to give some general principles for pruning instead of a specific pruning strategy. Unacceptable cyclic routes need to be well defined before pruning. Otherwise, we could just heavily penalize poor cyclic routes by defining cost metrics in the third stage (see Section 6). Pruning too aggressively may create an infeasible problem for the third stage. Let us get back to the example above. Due to the restrictions on drivers' working hours, long cyclic routes could be pruned even if they are the only ones which cover certain consignments. Then the third stage will never be

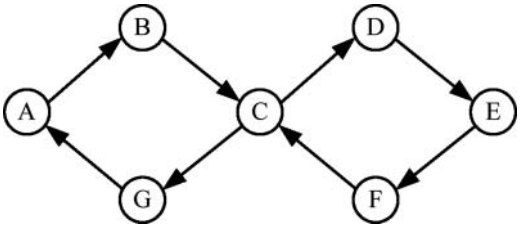

Figure 5. Two cyclic routes in the shape of '8'.

able to cover each arc of the service network. In practice, customers' demands come first and a LTL carrier will probably keep the poor cyclic route to guarantee a certain level of service. One possible way of satisfying the restriction is to have multiple drivers on one vehicle. The other drivers may have a break while one driver is driving (Goel & Gruhn 2006). Therefore, pruning can be done on the condition that it will not create an infeasible problem for the third stage.

6. Covering a service network

As mentioned in Section 3, we need to select a set of cyclic routes from a collection of acceptable cyclic routes generated in Sections 4 and 5. The selected cyclic routes should not only minimize the total cost but cover each arc in the solution network. Before that, the cost of each cyclic route should be defined first.

According to various requirements from different LTL carriers, metrics could be designed flexibly to measure the cost of each route. If a LTL carrier cares more about the distance, it could assign a value measuring the distance for each route. Then the objective is to keep the sum of values for each selected route to the minimum. This is common since longer total distance usually means more cost for a LTL carrier. Note that the route penalty here can also be a complex, nonlinear function of the distance or the travel time, which does not add further complexity of the set covering model. Other metrics such as the number of nodes may also be designed to suit different needs. For the purpose of testing our scheme, we designed three different metrics in Section 7. In this paper, we assume that each cyclic route has a single associated cost.

The problem can then be described as follows. There are three inputs. The first one is the complete set of arcs in a given solution network. The second one is a set of acceptable cyclic routes, each comprised of these arcs. In addition, each cyclic route has a cost. The objective is to select a subset of cyclic routes that minimize the total cost as well as covering each arc in the solution network.

Furthermore, the problem can be formulated as a weighted set covering problem using the following mathematical notations. Let $E = \{e_1, e_2, \cdots, e_m\}$ be the complete set of arcs. Each ground element in this set represents an arc in a given service network design solution. Let R represent the collection of all the acceptable cyclic routes. Each entry $r(r_1, r_2, \cdots, r_n)$ in set R denotes a cyclic route and is comprised of arcs in E, thus we have $r_1, r_2, \cdots, r_n \subseteq E$. Let p_r be the cost of cyclic route $r \in R$. The goal is to find a subset of R that has the minimum total cost to cover all of the elements of set E. Thus, the weighted set covering model can be given as follows:

$$\min \sum_{r \in R} z_r p_r \tag{10}$$

s.t.

$$\sum_{r \in R} z_r \delta_r^l \geq 1, \ \forall l = 1, 2, \cdots, m \tag{11}$$

The decision variable here is z_r, whose value indicates whether a cyclic route r is included or not. It is defined as follows:

$$z_r = \begin{cases} 1 & r \in R \text{ is chosen} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

The indicator δ_r^l shows whether an arc e_l is covered in a cyclic route r, as defined below:

$$\delta_r^l = \begin{cases} 1 & r \in R \text{ covers } e_l \in E \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

In the case where the cost of each route is equal, we set $p_r = 1$ for all possible values of r.

The resulting model is a binary (integer) program since each element of the decision vector is a 0/1 variable. As a state-of-the-art software package for integer programming, IBM ILOG CPLEX solver proves to be particularly powerful and appealing. In view of this, we utilize CPLEX to solve the problem. The computational results in Section 7 indicate its effectiveness.

7. Computational results and analysis

In order to evaluate the performance of the scheme we propose, experiments have been carried out based on the service network design solutions from Bai et al. (2012). Their solutions were obtained as results of experimental tests on a set of benchmark instances for service network design. Those instances were first generated by Crainic et al. (2000) and are identified with the letter C. In addition, each instance is characterized by five parameters. The first three are the number of nodes, arcs and commodities, respectively. The fourth parameter signifies whether the ratio of fixed costs over variable costs is high (F) or low (V), while the last one indicates whether the instance is tightly (T) or loosely (L) capacitated. Herein we selected solutions to the first eight instances as the input of our experiment, shown in Table 1. For each instance, the numbers of nodes and arcs in its solution are shown in the column entitled 'Size of Solution Networks'.

As the first stage of our scheme, the algorithm for finding all of the different cyclic routes was implemented in C++. Unless otherwise indicated, all of the computing was conducted on a laptop computer with a 2.13 GHz Intel Core i3 CPU and 2 Gigabyte of

Table 1. Computational results of generating all cyclic routes.

	Size of solution networks			
Instance	No. of nodes	No. of arcs	No. of cyclic routes	Searching time (s)
C20,230,200,V,L	20	57	8040	3.535
C20,230,200,F,L	20	46	968	1.857
C20,230,200,V,T	20	56	6254	2.649
C20,230,200,F,T	20	61	17,335	3.263
C20,300,200,V,L	20	66	114,856	8.968
C20,300,200,F,L	20	62	38,235	4.316
C20,300,200,V,T	20	71	167,128	17.388
C20,300,200,F,T	20	63	42,736	3.548

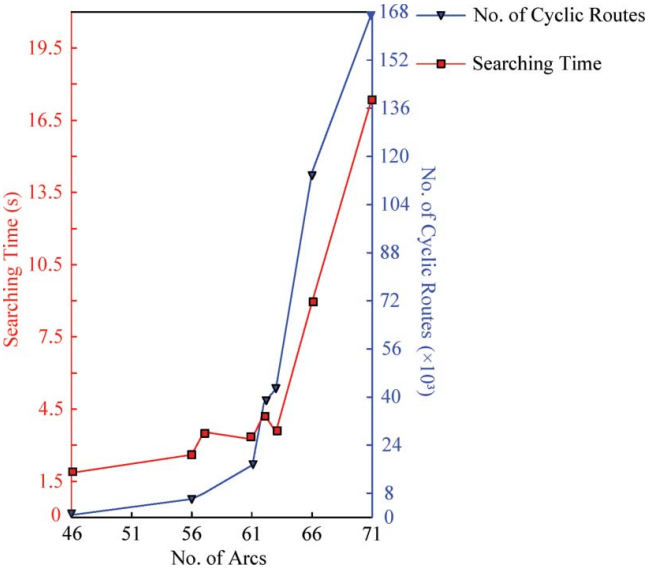

Figure 6. The rate of change for searching time and number of cyclic routes.

RAM, under a 32-Bit operating system. The total number of cyclic routes for each solution network, as well as the corresponding searching time, is indicated in Table 1. For solution networks with no more than 71 arcs, it is clear that the algorithm could efficiently find all cyclic routes, whose number amounts to 167,000, within 18 seconds.

Furthermore, we can see that the number of cyclic routes grows dramatically as the number of arcs increases. In order to visualize the rate of change, Figure 6 has been plotted. Despite possible small fluctuations, Figure 6 depicts an overall trend of roughly exponential growth in both the number of cyclic routes and searching time with the increase of the number of arcs. As a consequence, this poses a great challenge to large-scale solution networks in terms of memory and computing time. For instance, the number of cyclic routes climbs to 2,005,251 for instance named 'C30,520,100,V,L', whose solution network consists of 30 nodes and 91 arcs, with searching time going up to 251.316 seconds. Although our method is still effective for a problem on such scale, the time consumption may be unacceptable from the perspective of practical applications.

In order to measure the cost of each cyclic route, three metrics were designed in our experiment. In the case of 'equal cost', the cost of each cyclic route is equal. For convenience, the values of all of the costs are set to 1. The second metric derives its name 'counting cost' from counting the number of arcs in a cyclic route. The costs for the third metric, 'fixed cost', are obtained by adding together the cost of each arc included in a cyclic route. The cost of an arc here refers to the value of 'fixed_cost' for this arc, which can be extracted from the benchmark data-set described earlier. As an example, the fixed cost for each arc in the solution network of instance C20,230,200,V,L is shown in Table 2. This solution network consists of 57 arcs. Please note that the node representation of an arc in this table, let us say, $(i \rightarrow j)$ corresponds to an arc from node $(I + 1)$ to node $(j + 1)$ in the benchmark data-set. The original value of 'fixed_cost' for an arc is identified as 'old' in the table. To simplify future calculation, we multiplied the 'old' value by 0.01 and then

Table 2. The fixed cost for each arc in the solution network of instance C20,230,200,V,L.

	Fixed cost			Fixed cost			Fixed cost			Fixed cost	
Arc	old	new	Arc	old	new	Arc	old	new	Arc	old	new
0→2	1149	11	5→10	416	4	10→5	294	2	16→9	734	7
0→10	318	3	5→11	367	3	10→8	245	2	16→19	1369	13
0→13	367	3	5→16	1931	19	10→18	392	3	17→2	440	4
1→7	1491	14	6→0	1760	17	11→0	1638	16	17→7	245	2
1→19	1100	11	6→9	440	4	11→3	831	8	17→18	318	3
2→3	245	2	7→15	440	4	11→9	856	8	17→19	1320	13
2→4	538	5	7→16	489	4	11→17	978	9	18→1	1345	13
2→15	343	3	8→5	660	6	12→8	929	9	18→11	1296	12
3→2	343	3	8→12	367	3	12→11	660	6	19→0	294	2
3→4	294	2	8→14	612	6	13→5	294	2	19→1	1003	10
3→8	1907	19	8→19	1100	11	13→6	783	7	19→10	1198	11
3→13	367	3	9→5	783	7	14→3	269	2	19→17	807	8
4→11	563	5	9→12	563	5	14→8	1565	15			
4→17	880	8	9→14	1076	10	15→10	1467	14			
5→6	318	3	10→3	1785	17	15→17	2102	21			

truncated the resulting number by discarding the decimal part. The 'new' value for fixed cost was used for subsequent optimization instead of the 'old' value.

As stated in Section 5, good pruning depends on complete information from carriers and is better done case by case. Since such information is not included in the test data-set, we did pruning according to the cost of each route in the experiment. Overall, pruning was performed based on the principle that the routes, whose costs were close to the highest cost among all of the routes, were less likely to be in the optimal set. Hence, we had adequate reason to eliminate them without harming the third stage but we cannot guarantee it for all cases.

For the counting cost, we chose as the pruning criterion the rule that any route whose cost value was greater than 14 was to be pruned. There were other alternative values possible and we chose the value 14 for demonstration purposes only. Note that each cyclic route theoretically has 20 arcs at most because every instance in the experiment consists of 20 nodes. Regarding the fixed cost, cost tables like Table II are not the same for different instances. Consequently, the instances range in the costs of routes from relatively low to high. To deal with this, we chose for each instance a value out of three options, namely 85, 135 and 185. These three values corresponded to the case of low, medium and high costs, respectively, with a constant difference between the consecutive numbers. The pruning criterion for the case of low costs was then defined as removing those routes whose cost values are greater than 85. In the case of medium and high costs, the pruning criteria were described in a similar fashion. With respect to the equal cost, it stands to reason that a good solution tends to include as many arcs as possible in one route. In this way, all of the arcs could be covered with fewer routes and thus a lower cost is induced. For this reason, we chose to prune the routes which contained fewer than 11 arcs. The results of pruning under three different conditions are shown in Table 3.

It can be seen from Table 3 that each instance contains poor cyclic routes that can be pruned. We shall see later that more than 80% of this pruning does not cause any change of the final solutions in the third stage. With an appropriate pruning criterion, Table 3 also indicates that a large proportion of routes can be pruned for most instances. In the case of fixed cost, the average percentage of pruned routes is more than 35%, whereas the average percentages in the cases of counting cost and equal cost are 32% and 16%,

Table 3. Results of pruning under three different conditions.

	Equal cost			Counting cost			Fixed cost		
Instance	Pruning criterion (no. of Arcs <)	No. of pruned routes	Percentage of pruned routes (%)	Pruning criterion (cost value >)	No. of pruned routes	Percentage of pruned routes (%)	Pruning criterion (cost value >)	No. of pruned routes	Percentage of pruned routes (%)
C20,230,200,V,L	11	1304	16.22	14	2743	34.11	135	813	10.11
C20,230,200,F,L	11	347	35.85	14	120	12.40	185	580	59.92
C20,230,200,V,T	11	1715	27.42	14	782	12.50	85	2346	37.51
C20,230,200,F,T	11	3149	18.17	14	4584	26.44	185	6272	36.18
C20,300,200,V,L	11	6411	5.58	14	61,644	53.67	85	26,898	23.42
C20,300,200,F,L	11	4360	11.40	14	14,743	38.56	135	25,783	67.43
C20,300,200,V,T	11	11,960	7.16	14	76,927	46.03	85	6204	3.71
C20,300,200,F,T	11	4782	11.19	14	16,171	37.84	135	20,702	48.44

respectively. We can expect a higher average percentage for the latter two cases if the pruning criterion is tailor-made for each instance. Pruning in this fashion helps to greatly reduce the memory usage and the risk of out-of-memory, because there are usually huge amounts of information about the cyclic routes that need to be stored as the inputs for the third stage. Besides, a lower-level memory requirement enables the proposed algorithm to deal with larger sized instances. Pruning is further discussed in the results of the third stage.

Now we come to the third stage of our scheme, namely selecting a set of cyclic routes, from all of the cyclic routes, to cover the entire network. The weighted set covering model established in Section 6 was solved with IBM ILOG CPLEX V12.6.1. The input data of this model were created based on the arcs and cyclic routes obtained in the first and second stages, respectively. The optimization results for three different kinds of costs and corresponding running times are reported in Table 4.

As can be seen from Table 4, CPLEX could find optimal solutions to all of the eight instances within 68 seconds. In particular, the running time for each instance reduced noticeably after pruning. More importantly, more than 80% of this pruning does not cause any change of the final solutions. There are only four cells where the final solution failed to attain the optimal value or the same set of optimal routes after pruning. In fact, this failure is a direct result of overly aggressive pruning. Take the instance 'C20,230,200,

Table 4. Optimization results of selecting cyclic routes under three different conditions.

	Equal cost			Counting cost			Fixed cost		
		Time (s)			Time (s)			Time (s)	
Instance	Optimal value of objective function	Before pruning	After pruning	Optimal value of objective function	Before pruning	After pruning	Optimal value of objective function	Before pruning	After pruning
C20,230,200,V,L	4	4.92	2.98	57	2.95	0.92	437	5.47	1.81
C20,230,200,F,L	5	0.85	*	46	0.66	0.08	795	0.92	0.38
C20,230,200,V,T	6	2.93	**	56	1.76	0.86	358	3.89	1.12
C20,230,200,F,T	6	6.67	3.89	61	3.12	**	775	4.50	1.25
C20,300,200,V,L	4	59.47	38.39	66	67.09	**	327	30.85	16.58
C20,300,200,F,L	5	15.09	7.96	62	11.61	6.07	678	9.43	4.35
C20,300,200,V,T	6	64.92	39.22	71	66.08	32.84	301	48.26	21.70
C20,300,200,F,T	5	13.01	8.28	63	15.33	6.46	603	10.60	5.07

*: The objective function failed to attain the optimal value after pruning.
**: The optimal objective value was attained but the actual solution (i.e. the set of optimal routes) was different.

Table 5. The optimal set of cyclic routes for instance C20,230,200,V,L with equal cost.

Index of a cyclic route	Node representation	No. of arcs	Equal cost
1511	0→13→5→6→9→14→8→19→10 →3→2→15→17→18→11→0	15	1
2538	0→10→5→16→19→17→2→4→11 →9→12→8→14→3→13→6→0	16	1
5745	0→2→3→8→5→11→17→7→15 →10→18→1→19→0	13	1
7337	10→8→12→11→3→4→17→19→1 →7→16→9→5→10	13	1

F,T' as an example. Its optimal solution includes a cyclic route whose cost value is 15. In the case of counting cost, any route whose cost value is greater than 14 will be pruned according to the pruning criterion. Hence, this cyclic route will be pruned before the third stage. If the pruning criterion were set to be 15 or higher, the final solution would still attain the same set of optimal routes after pruning. Conversely, sometimes pruning was done too cautiously. Take the instance 'C20,300,200,V,T' with fixed cost as an example. It is shown in Table 3 that only 3.71% of the routes were pruned. If the pruning criterion were set to be 60, the number of pruned routes would increase to 72,067 and the percentage would rise to 43.12. This result would be more desirable since the optimal solution still remained unchanged. It is hard to define an appropriate pruning criterion suitable for all cases. Part of the problem here is, of course, that we utilized only example values for the pruning here, whereas in real situations users will probably have more information about what is likely to happen and what is acceptable and could use this to tune the pruning. In order to achieve a desirable result, the pruning criterion should probably be tailored for each instance, using knowledge about the problem to solve, or investigating alternatives.

To explain these optimization results in further detail, we think about instance C20,230,200,V,L as an example. As stated above, its solution network consists of 57 arcs. In the case of equal cost, the optimal set of cyclic routes is displayed in Table 5. There are 4 cyclic routes selected to cover all of the 57 arcs. Consequently, the optimum of the objective function should be 4, which is exactly the corresponding 'Optimal value of objective function' in Table 4. In the context of counting cost, the optimal set includes 13 cyclic routes, as shown in Table 6. Since the cost of a route refers to the number of arcs in it, the total cost comes to 57. The sum explains where

Table 6. The optimal set of cyclic routes for instance C20,230,200,V,L with counting cost.

Index of a cyclic route	Node representation	No. of arcs	Counting cost
1768	0→13→5→6→0	4	4
1896	0→10→18→11→0	4	4
4117	0→2→15→10→5→11→3 →13→6→9→12→8→19→0	13	13
6658	2→4→17→2	3	3
7042	2→3→2	2	2
7296	10→8→12→11→17→19→10	6	6
7789	10→3→8→5→10	4	4
7921	1→19→17→18→1	4	4
7924	1→7→16→19→1	4	4
8000	7→15→17→7	3	3
8023	3→4→11→9→14→3	5	5
8026	8→14→8	2	2
8038	5→16→9→5	3	3

Table 7. The optimal set of cyclic routes for instance C20,230,200,V,L with fixed cost.

Index of a cyclic route	Node representation	No. of arcs	Fixed cost
1230	0→13→5→11→0	4	24
2964	0→10→3→8→19→0	5	52
4789	0→2→4→17→18→1→7→15→10 →8→12→11→9→14→3→13→6→0	17	130
6196	2→15→17→2	3	28
7042	2→3→2	2	5
7043	10→18→11→17→19→10	5	48
7547	10→5→10	2	6
7922	1→19→1	2	21
7978	7→16→19→17→7	4	27
8025	3→4→11→3	3	15
8026	8→14→8	2	21
8030	8→5→16→9→12→8	5	46
8039	5→6→9→5	3	14

the value of 57 in Table 4 comes from. In respect of fixed cost, 13 cyclic routes depicted in Table 7 comprise the optimal set. As the fixed cost of each arc can be looked up in Table 2, we can calculate the sum of each route's cost. That is how we get 437 for the 'Optimal value of objective function' in Table 4.

Summarizing all of the computational results above, we can see that the proposed scheme could efficiently find the optimal set of cyclic routes to cover a given service network including no more than 71 arcs within 86 seconds. This result indicates not only the effectiveness of the proposed scheme, but also its efficiency to transform the optimized service network into a set of cyclic routes.

As is known, the SNDP is NP-hard and there is no efficient solution procedure for large-scale networks. We have to admit that when the scale of the service network design solution grows much larger, our scheme is confronted with a similar problem. Its performance becomes poor due to unacceptable time consumption and memory requirements. In other words, without modification, our scheme is not currently applicable for large-scale problems either.

8. Conclusions and future directions

To assign a cyclic route to each vehicle and keep services running well for LTL carriers, we propose to transform the solution network of the Arc-based SNDP model into a set of cyclic routes that cover all arcs in the network. The solution strategy is decomposed into three stages. The first stage aims to find all of the different cyclic routes in a solution network and we present an algorithm based on depth-first search to solve it. One great advantage of the decomposition strategy is that it enables us to prune poor cyclic routes before route assignment. Pruning can also be done in the first stage to make the proposed algorithm faster. The third stage is to select a set of cyclic routes to cover the entire network. It is formulated as a weighted set covering problem. The resulting model as an integer program is solved with CPLEX solver and optimal solutions are obtained in reasonable time. Experimental results have shown that the proposed scheme is quite effective with regard to solution quality and computational efficiency.

As we have mentioned, one limitation of the proposed strategy is that it is not applicable to larger problems. This is not surprising if we consider the fact that generally neither

SNDP nor other large-scale NP-hard integer programs can be solved to optimality within reasonable time. Future progress in network design and integer programming is expected to help improve the performance of our scheme on medium and large-scale problems. Various heuristic optimizations in the choice of cycles to use, or column generation approaches are obvious candidates. One aim of this paper is to encourage other researchers to consider this interesting problem.

We take vehicle balancing constraints into account in this paper. As a matter of fact, there are many other issues for LTL carriers to consider when setting up transport service plans. For instance, working time regulations are of extraordinary importance. LTL carriers must organize the work of drivers in a way that drivers are able to comply with the respective regulations. Therefore, it will be of value to introduce time constraints into the problem for practical transport planning in the future. This three-phase scheme is ideal for this, since many cycle-length regulations can be considered in the route generation and cost function design.

Acknowledgments

This work was supported by the International Doctoral Innovation Centre (IDIC) scholarship scheme. We also greatly acknowledge the support from Ningbo Education Bureau, Ningbo Science and Technology Bureau, China's MOST and The University of Nottingham.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This work is supported by the National Natural Science Foundation of China [grant number NSFC 71471092]; Zhejiang Natural Science Foundation [grant number LR17G010001]; Ningbo Science and Technology Bureau [grant number 2011B81006], [grant number 2014A35006].

ORCID

Xiaoping Jiang http://orcid.org/0000-0002-3588-1675
Jason Atkin http://orcid.org/0000-0002-7187-4982
Graham Kendall http://orcid.org/0000-0003-2006-5103

References

Ahuja RK, Magnanti TL, Orlin JB. 1993. Network flows: theory, algorithms, and applications. Upper Saddle River (NJ): Prentice-Hall.

Andersen J, Christiansen M, Crainic TG, Gronhaug R. 2011. Branch and price for service network design with asset management constraints. Transp Sci. 45:33–49.

Andersen J, Crainic TG, Christiansen M. 2009. Service network design with management and coordination of multiple fleets. Transp Res. 17:197–207.

Azadeha A, Farahani MH, Eivazy H, Shirkouhi SN, Asadipour G. 2013. A hybrid meta-heuristic algorithm for optimization of crew scheduling. Appl Soft Comput. 13:158–164.

Bai R, Kendall G, Li J. 2010. An efficient guided local search approach for service network design problem with asset balancing. In: Proceedings of the International Conference on Logistics Systems and Intelligent Management; 2010 Jan 9–10; Harbin (China): IEEE; p. 110–115.
Bai R, Kendall G, Qu R, Atkin J. 2012. Tabu assisted guided local search approaches for freight service network design. Inform Sci. 189:266–281.
Bai R, Wallace SW, Li J, Chong AY-L. 2014. Stochastic service network design with rerouting. Transp Res. 60:50–65.
Bai R, Xue N, Chen J, Roberts GW. 2015. A set-covering model for a bidirectional multi-shift full truckload vehicle routing problem. Transp Res. 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. 46:367–383.
Benslimane MT, Benadada Y. 2013. Ant colony algorithm for the multi-depot vehicle routing problem in large quantities by a heterogeneous fleet of vehicles. INFOR. 51:31–40.
Berbeglia G, Cordeau J-F, Gribkovskaia I, Laporte G. 2007. Static pickup and delivery problems: a classification scheme and survey. TOP. 15:1–31.
Caprara A, Fischetti M, Toth P. 1999. A heuristic method for the set covering problem. Oper Res. 47:730–743.
Cordeau J-F, Laporte G. 2003. The dial-a-ride problem (DARP): variants, modeling issues and algorithms. 4OR-Q J Oper Res. 1:89–101.
Cordeau J-F, Laporte G, Ropke S. 2008. Recent models and algorithms for one-to-one pickup and delivery problems. In: Golden B, Raghavan S, Wasil E, editors. The vehicle routing problem: latest advances and new challenges. New York (NY): Springer; p. 327–357.
Costa AM. 2005. A survey on benders decomposition applied to fixed-charge network design problems. Comput Oper Res. 32:1429–1450.
Crainic TG. 2000. Service network design in freight transportation. Eur J Oper Res. 122:272–288.
Crainic TG, Gendreau M, Farvolden JM. 2000. A simplex-based tabu search method for capacitated network design. INFORMS J Comput. 12:223–236.
Crainic TG, Hewitt M, Toulouse M, Vu DM. 2016. Service network design with resource constraints. Transp Sci. 50:1380–1393.
Crainic TG, Li Y. 2006. A first multilevel cooperative algorithm for capacitated multicommodity network design. Comput Oper Res. 33:2602–2622.
Erera A, Karacık B, Savelsbergh M. 2008. A dynamic driver management scheme for less-than-truckload carriers. Comput Oper Res. 35:3397–3411.
Ghamlouche I, Crainic TG, Gendreau M. 2004. Path relinking, cycle-based neighbourhoods and capacitated multicommodity network design. Ann Oper Res. 131:109–133.
Goel A, Gruhn V. 2006. Drivers' working hours in vehicle routing and scheduling. In: Proceedings of the IEEE Intelligent Transportation Systems Conference; 2006 Sept 17–20; Toronto (Canada): IEEE; p. 1280–1285.
Groiez M, Desaulniers G, Marcotte O. 2014. Valid inequalities and separation algorithms for the set partitioning problem. INFOR. 52:185–196.
Kok AL, Meyer CM, Kopfer H, Schutten JMJ. 2010. A dynamic programming heuristic for the vehicle routing problem with time windows and European community social legislation. Transp Sci. 44:442–454.
Laporte G. 1992. The vehicle routing problem: an overview of exact and approximate algorithms. Eur J Oper Res. 59:345–358.
Pederson MB, Crainic TG, Madsen OBG. 2008. Models and tabu search metaheuristics for service network design with asset-balance requirements. Transp Sci. 43:158–177.
Rancourt M-E, Cordeau J-F, Laporte G. 2012. Long-haul vehicle routing and scheduling with working hour rules. Transp Sci. 47:81–107.
Smilowitz KR, Atamtürk A, Daganzo CF. 2003. Deferred item and vehicle routing within integrated networks. Transp Res. 39:305–323.
Spoorendonk S, Desaulniers G. 2010. Clique inequalities applied to the vehicle routing problem with time windows. INFOR. 48:53–67.

Sundar S, Singh A. 2012. A hybrid heuristic for the set covering problem. Oper Res. 12:345–365.

Tarjan RE. 1972. Depth-first search and linear graph algorithms. SIAM J Comput. 1:146–160.

Teypaz N, Schrenk S, Cung VD. 2010. A decomposition scheme for large-scale service network design with asset management. Transp Res. 46:156–170.

Thapalia BK, Crainic TG, Kaut M, Wallace SW. 2011. Single-commodity stochastic network design with multiple sources and sinks. INFOR. 49:193–211.

Vu DM, Crainic TG, Toulouse M. 2013. A three-phase matheuristic for capacitated multi-commodity fixed-cost network design with design-balance constraints. J Heuristics. 19:757–795.

Weiss MA. 2013. Data structures and algorithm analysis in C++. 4th ed. Upper Saddle River (NJ): Pearson Education.

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