Bai, Ruibin and Kendall, Graham and Qu, Rong and Atkin, Jason A.D. (2012) Tabu assisted guided local search approaches for freight service network design. Information Sciences, 189 . pp. 266-281. ISSN 1872-6291

Tabu assisted guided local search approaches for freight service network design

Ruibin Bai[a,*], Graham Kendall[b], Rong Qu[b], Jason Atkin[b]

[a]*Division of Computer Science, University of Nottingham Ningbo China.*
Tel.: +86-574-88180278, Fax: +86-574-88180125
[b]*School of Computer Science, University of Nottingham, Nottingham, UK.*

Abstract

The service network design problem (SNDP) is a core problem in freight transportation. It involves the determination of the most cost-effective transportation network and the characteristics of the corresponding services, subject to various constraints. The scale of the problem in real-world applications is usually very large, especially when the network contains both the geographical information and the temporal constraints which are necessary for modelling multiple service-classes and dynamic events. The development of time-efficient algorithms for this problem is, therefore, crucial for successful real-world applications. Earlier research indicated that guided local search (GLS) was a promising solution method for this problem. One of the advantages of GLS is that it makes use of both the information collected during the search as well as any special structures which are present in solutions. Building upon earlier research, this paper carries out in-depth investigations into several mechanisms that could potentially speed up the GLS algorithm for the SNDP. Specifically, the mechanisms that we have looked at in this paper include a tabu list (as used by tabu search), short-term memory, and an aspiration criterion. An efficient hybrid algorithm for the SNDP is then proposed, based upon the results of these experiments. The algorithm combines a tabu list within a multi-start GLS approach, with an efficient feasibility-repairing heuristic. Experimental tests on a set of 24 well-known service network design benchmark instances have shown that the proposed algorithm is superior to a previously proposed tabu search method, reducing the computation time by over a third. In addition, we also show that far better results can be obtained when a faster linear program solver

*Corresponding author
Email addresses: `ruibin.bai@nottingham.edu.cn` (Ruibin Bai), `gxk@cs.nott.ac.uk` (Graham Kendall), `rxq@cs.nott.ac.uk` (Rong Qu), `jaa@cs.nott.ac.uk` (Jason Atkin)

is adopted for the sub-problem solution. The contribution of this paper is an efficient algorithm, along with detailed analyses of effective mechanisms which can help to increase the speed of the GLS algorithm for the SNDP.

Keywords:
Logistics, Freight Transportation, Guided Local Search, Service Network Design, Linear Programming

---

1. Introduction

The service network design problem (SNDP) involves the determination of the most cost-effective transportation network and the services which it will provide. Solutions must satisfy both geographical and temporal constraints, reflecting the demands of customers, network availability and capacity, and transport fleet assignments. Recent advances in service network design are playing a significant role in the development of the next generation freight transportation systems. These systems automate the design and operation of transportation networks, providing solutions that not only allocate expensive resources in an optimal (or near optimal) fashion but are also able to cope with the uncertain events which happen in the real world, such as disturbances in demands, traffic congestion, and vehicle breakdowns.

There have been only limited reports of successful real-life SNDP applications in the scientific literature. These include the early work by Crainic and Rousseau [16] and later work by Armacost et al.[5] and Jansen et al. [20]. One of the common features of the previous work has been the static nature of the test problem instances, in the sense that all of the data which captures the characteristics of the SNDP problem has been assumed to be unchanging over time. In addition, a significant computational time was also permitted in previous work, allowing methods to achieve higher quality solutions. Although reasonable for an off-line solution method where the problem only has to be solved once, these computational times are perhaps unrealistic given the dynamic on-line nature of the real world problems. In a more realistic scenario, a planner has to solve SNDP more frequently, needing to obtain a solution very quickly, potentially even in real-time, to cope with changes as they happen, thus the computational times can become an impediment for the adoption of these methods. Computational time has remained a major issue for the last twenty years or so in the development of solution methods for large-scale service network design, despite rapid developments in computing power.

Figure 1 illustrates a typical SNDP solving process in an uncertain environment. At each planning horizon, the solver takes as input the schedules which were determined in the previous period, either as a fixed partial solution or as additional constraints for the problem. Since the solver has to not only deal with dynamic events as they happen, but also take into account possible future events or demands (so that the schedule generated by the SNDP solver is not entirely myopic), estimations have to be provided for uncertain data along with the values of known data.

The SNDP solver has to be reapplied to handle the changes whenever a dynamic event occurrence invalidates the previous estimation to the point where the quality of the current solution is compromised, or the solution becomes infeasible. Unfortunately, good estimations for uncertain data are not always possible, but poor estimations may lead to frequent re-application of the SNDP solver. For this reason, it is essential that the SNDP solver has the capability to find high quality solutions very quickly.

Armacost et al. [5] reported a successful static SNDP application for the UPS `Next-Day` delivery service by exploiting special problem structures. However, it is questionable whether it would be suitable for applications with `multiple-class` freight services (for example first-class, second-class, and deferred services), where the planning has to foresee future demands. Integrating services of multiple classes would probably require the introduction of a `time-space network`, which would dramatically increase the problem sizes [3], and hence the solution times. Problems of such large sizes are very difficult for current approaches. For example, Pedersen et al.'s approach [24] required an hour of computation time on a PC with a relatively high computing power (2.26GHz CPU), even for instances which are only small or medium sized (30 nodes, 700 arcs and 400 commodities) from an industrial point of view.

This current state of the art in service network optimisation has motivated us to develop new algorithms and mechanisms that 1) could considerably speed up the solution methods for problems with a dynamic nature, without deteriorating the solution quality; 2) are able to tackle static problems of a much larger size with far lower computational times than current methods require. The research in this paper aims to contribute towards the development of more realistic real-world SNDP applications in the freight transportation industry. In particular, it builds upon the initial success of a guided local search metaheuristic [7] for fast and competitive SNDP solutions and extends the work in [6] by describing the experimentation and subsequent analysis

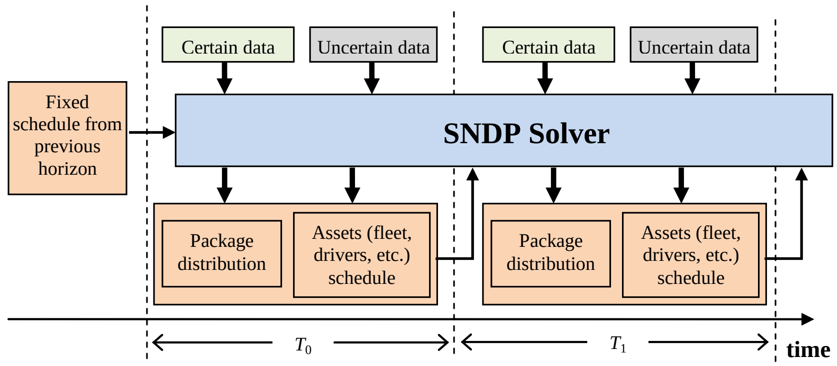


Figure 1: Service network design process in uncertain environments.

which was performed in order to identify and test the elements and mechanisms which could further improve the algorithmic performance.

The remainder of the paper is structured as follows: section 2 provides a brief introduction of the problem and an overview of the research so far in freight service network design. Section 3 presents the well known arc-node formulation for SNDP and section 4 contains details of the application of a basic GLS for the service network design problem. Section 5 is the main focus of the paper, containing experimental analysis of various mechanisms that could affect the performance of the GLS algorithm for the SNDP. Section 6 studies how a better linear program solver contributes to the performance of the algorithm. Section 7 concludes the paper and identifies future research directions.

2. Problem Description and Literature Review

The service network design problem (SNDP) is an important tactical/operational freight transportation planning problem. It is of particular interest for less-than-truckload (LTL) transportation and express delivery services, where consolidation of deliveries is widely adopted in order to maximise the utilisation of freight resources [12]. The problem is usually concerned with finding a cost-minimising transportation network configuration that satisfies the delivery requirements for all of the commodities, each of which is defined by a source node, target node, and quantity of demand. More specifically, the service network design problem involves the search for optimal decisions in terms of the service characteristics (for example, the selection of routes to utilise 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 [27].

The service network design problem is similar to the capacitated multicommodity network design (CMND) problem except that the SNDP has an extra degree of complexity due to the required *balance constraint* for freight assets (for example, ensuring that vehicle routes are contiguous and that vehicles are in the correct positions after each planning cycle), which does not apply to the standard CMND. Both the CMND and the SNDP are known to be NP-Hard problems [18]. The remainder of this section provides a brief overview of the previous research into service network design. More comprehensive reviews can be found in [12, 15, 27].

Service network design is closely related to the classic network flow problems [1]. Early work in this field includes [16, 25, 17]. Crainic et al. [14] applied a tabu search metaheuristic to the container allocation/positioning problem and Crainic et al. [13] investigated a hybrid approach for CMND, combining a tabu search method with pivot-like neighbourhood moves and column generation. Ghamlouche et al. [18] continued the work and proposed a more efficient cycle-based neighbourhood structure for CMND. Experimental tests, within a simple tabu search framework, demonstrated the superiority of the method to the earlier pivot-like neighbourhood moves in [13]. This approach was later enhanced by adopting a path-relinking mechanism [19].

Barnhart and her research team [9, 21] addressed a real-life air cargo express delivery service network design problem. The problem is characterised by its large problem sizes and the addition of further complex constraints to those which are in existence in the general SNDP model. A `tree formulation` was introduced and the problem was solved heuristically using a method based upon column generation. Armacost et al. [4] introduced a new mathematical model based on an innovative concept called the `composite variable`, which has a better LP 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 the model to other freight transportation applications, especially when there are several classes of services being planned simultaneously.

Recent work by Pederson et al. [24] studied more generic service network design models in which the asset balance constraint was present. A multi-start metaheuristic, based on tabu search, was developed and tested on a set of benchmark instances. The tabu search method outperformed a commercial MIP solver when computational time was limited to one hour per instance on a test PC with a Pentium IV 2.26GHz CPU. Andersen et al. [3] compared the node-arc based formulation, the path-based formulation and a cycle-based formulation for SNDP problems. Computational results on a set of small randomly generated instances indicated that the cycle-based formulation gave significantly stronger bounds than the other two and hence may allow for much faster solution of problems. More recent work by Bai et al. [7] attempted to further reduce the computational time and investigated a guided local search approach. The computational study, based on a set of popular benchmark instances, showed that the guided local search, if configured appropriately, was able to obtain solutions of a similar quality level to the tabu search but with only two thirds of the computational time, even when executed on a slightly slower machine. Barcos et al. [8] investigated an ant colony optimization (ACO) approach to address a variant (simplified) freight service network design problem. The algorithm was able to obtain solutions better than those adopted in the real-world within a reasonable computational time. Andersen et al. [2] studied a branch and price method for the service network design problem. Although the proposed algorithm was able to find solutions of higher quality than the previous methods, the 10-hour computational time required by the algorithm poses a great challenge for its practical applications. Chiou [11] proposed a two-level mathematical programming model for the logistics network design. The upper level model is concerned with optimising the network configuration while the lower level optimises the flow distribution with flow-dependent marginal costs. However, the model does not take into account the asset balance constraints.

The research mentioned above primarily dealt with problems of a static nature. However, service network planning involves several uncertain aspects, such as unpredictable demands, traffic congestion, delays, and vehicle breakdowns. Optimal solutions for a static problem may turn out to have poor quality or even lose feasibility as a result of the unpredictable dynamic events. Liu et al. [22] proposed a two-stage approach based on stochastic programming to model the interdependencies between transportation assets and potential uncertainties. Bock [10] proposed a dynamic scheduling like approach to deal with uncertainties. From an initial plan, which was generated using estimated data, the system dynamically re-solved the current plan in order to adapt to the evolving problem, so the SNDP had to be solved repeatedly. Due to the lack of predictability for the data, we have adopted the latter type of approach, focusing upon speed of execution to allow the algorithm to be re-executed as the situation changes, however introducing some elements of stochastic programming would be an interesting area for future research.

The goal of this paper is to develop a much more efficient algorithm which could be utilised in conjunction with existing technologies, such as parallel computing, to allow the solution of much larger scale SNDP instances (of the type which often occur when using the time-space network formulation) or of dynamic SNDP problems where the computational time is critical. This paper contributes to the literature not only a more efficient hybrid metaheuristic approach for the SNDP, but also, more importantly, an experimental evaluation of the behaviour and performance of several effective components and mechanisms within a GLS framework. We expect that these findings will also be useful for other researchers working on similar algorithms.

3. Freight Service Network Design Problem (SNDP) Model

Table 1: List of notations used in the SNDP model

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

We focus on a specific recently studied service network design formulation which was described in [24] and which we also present here for completeness. A summary list of the notations used in the model is provided in Table 1 and the model is discussed below.

Let $\mathscr{G} = (\mathscr{N}, \mathscr{A})$ denote a directed graph with nodes $\mathscr{N}$ and arcs $\mathscr{A}$. Let (i, j) denote the arc from node i to node j. Let $\mathscr{K}$ be the set of commodities. For each commodity $k \in \mathscr{K}$, 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 otherwise. Let x^k_{ij} 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^k_{ij} denote the variable cost of moving one unit of commodity k along arc (i, j).

The service network design problem can then be formulated as follows:

minimise

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

subject to

$$\begin{aligned} \sum_{k\in\mathscr{K}} x^k_{ij} &\leq u_{ij}y_{ij} \quad \forall (i,j) \in \mathscr{A} && (2)\\ \sum_{j\in\mathscr{N}^+(i)} x^k_{ij} - \sum_{j\in\mathscr{N}^-(i)} x^k_{ji} &= b^k_i, \quad \forall i \in \mathscr{N}, \forall k \in \mathscr{K} && (3)\\ \sum_{j\in\mathscr{N}^-(i)} y_{ji} - \sum_{j\in\mathscr{N}^+(i)} y_{ij} &= 0 \quad \forall i \in \mathscr{N} && (4) \end{aligned}$$

where $x^k_{ij} \geq 0$ and $y_{ij} \in \{0,1\}$ are the decision variables. The network capacity constraint (2) ensures that the maximum flow along each arc (i, j) is limited by the arc capacity. The flow conservation constraint (3) ensures that the entire flow of each commodity is delivered to its destination, where $\mathscr{N}^+(i)$ denotes the set of outward neighbours of node i and $\mathscr{N}^-(i)$ the set of inward neighbours. b^k_i is the outward flow of commodity k for node i, so we set $b^k_i = d^k$ if $i = o(k)$, $b^k_i = -d^k$ if $i = s(k)$, and $b^k_i = 0$ otherwise. Constraint (4) is the *asset-balance constraint*, which is missing from the standard CMND formulation, as previously discussed, and which ensures the balance of transportation assets (i.e. vehicles) at the end of each planning period.

For a given set of design variables $\bar{\mathbf{y}} =< \overline{y_{00}}, ..., \overline{y_{ij}}, ... >$, 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 $\overline{\mathscr{A}}$ denote the set of open arcs in the design vector $\bar{\mathbf{y}}$, then flow distribution variables (x_{ij}^k) for all open arcs ($(i,j) \in \overline{\mathscr{A}}$) can be obtained by solving the following capacitated multicommodity min-cost flow problem (CMMCF), where $x_{ij}^k \geq 0 \; \forall (i,j) \in \overline{\mathscr{A}}, k$:

minimise

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

subject to

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

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

It was shown in [7] that a multi-start guided local search (GLS) approach performed well on this problem, producing results which were competitive with a recently proposed tabu search method [24], but in a much lower computational time. Based upon this initial success, this research aims to investigate, in detail, what contributed to this success and whether there are components and mechanisms that may lead to further improvement either in terms of computational time or solution quality. In particular, we have investigated: a) how effectively the current GLS explores the search space rather than getting stuck in a locally optimal set of solutions; b) whether more efficient mechanisms can be found and integrated within GLS; c) other factors which could potentially reduce the search time.

4. Guided Local Search For SNDP

4.1. Guided local search

Guided local search (GLS) is a metaheuristic which was designed for constraint satisfaction and combinatorial optimisation problems [26]. Like tabu search, GLS makes use of information gathered during the search to guide it and enable it to escape locally optimal regions rather than cycling between a few locally good solutions. In addition, GLS also exploits domain knowledge by penalising "unpopular" features in a candidate solution. The core of the guided local search method is the identification of a set of features and the determination of a transformed evaluation function. For a given solution s, the transformed evaluation function will have the following form:

$$E(s) = g(s) + GLS\_pen = g(s) + \lambda \times \sum_r (p_r \times I_r(s)) \tag{8}$$

where $g(s)$ is the original objective function. The formulations $g(s)$ and $g(\mathbf{x},\mathbf{y})$ are used interchangeably in this paper, since function g could be expressed in terms of the entire solution s or in terms of the vectors $\mathbf{x}$ and $\mathbf{y}$ of decision variables, as in Equation (9).

The variable p_r is the current penalty for the presence 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. GLS_pen is, therefore, the scaled (by λ) penalty summation which is applied by the GLS. λ is a control parameter which is often estimated by $\lambda = \alpha \times g(s^*)/\sum_r I_r(s^*)$ where s^* is the current best solution and α is a parameter that is less problem-dependent than λ.

The penalty value, p_r, for each feature, r, can be dynamically changed, if desired. The selection of features to be penalised in the GLS is based upon a utility value $util_r(s)$ for the feature r, which is defined by $util_r(s) = I_r(s) \times h_r/(1+p_r)$, where h_r is a cost associated with feature r. Given these definitions, the basic GLS approach can be illustrated by Figure 2.

---

input: an initial solution s_0, an original objective function
$g(s)$, a set of features R, the cost h_r associated with each
feature $r \in R$ and a scaling parameter λ.
begin
    foreach $r \in R$, set $p_r := 0$;
    $E(s) = g(s) + \lambda \times \sum_r p_r I_r(s)$;
    while stopping criterion is false
        $s \leftarrow LocalSearch(s, E(s))$
        foreach $r \in R$ **do**
            $util_r(s) = I_r(s) \times \frac{h_r}{1+p_r}$
            Find r with maximum $util_r$, set $p_r{+}{+}$;
        end foreach
    end while
    return $s^* \leftarrow$ best solution found according to function $g(s)$;
end

Figure 2: Pseudo-code for a basic guided local search procedure

It can be seen from Figure 2 that, unlike many metaheuristics, guided local search not only makes use of historical information from the search but also provides more flexibility for an algorithm designer to exploit special structures of the problem in terms of solution features and their associated costs. Therefore, if appropriate solution features can be determined and used, GLS can converge to a high quality solution more quickly than other metaheuristics (such as simple tabu search or simulated annealing). This may explain why GLS needed only two thirds of the computation time of the tabu search method as shown in [7].

In order to analyse and understand how each component or mechanism contributes to the performance of GLS, we started from a basic implementation of the GLS, from which we experimented and investigated various alternatives to improve its performance.

4.2. A basic GLS implementation for the SNDP

To apply a basic GLS to the SNDP, a number of issues had to be addressed. These are explained in this section.

4.2.1. Evaluation Function and Constraint Handling

To compute the evaluation function (8), one needs to identify a set of features and their associated costs. 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 $r \in \mathscr{A}$. An alternative choice of feature cost could take into account both the fixed cost, the variable cost and the popularity of the arc, however, this would inevitably introduce further parameters into the algorithm so was not considered in this paper. The network capacity constraint (2) and the flow conservation constraint (3) are handled directly in the local search procedure, so that any moves which violate any of these constraints will be discarded. However, the asset-balance constraint (4) is relaxed so that violations of this constraint are permitted, but are penalised according to the following penalty function:

$$g(\mathbf{x},\mathbf{y}) = z(\mathbf{x},\mathbf{y}) + Infeas = z(\mathbf{x},\mathbf{y}) + \tau \times \overline{f} \times \sum_{i \in \mathscr{N}} |\psi_i|^{\gamma} \tag{9}$$

where $Infeas$ is a measurement of the infeasibility of the solution s in terms of constraint (4), $\overline{f}$ is a scaling factor which is designed to give greater network independence and is defined as the average of the fixed costs of the arcs in the network, and ψ_i, the *node asset-imbalance*, denotes the difference (or imbalance) between outgoing open arcs and incoming open arcs at node i, as expressed by Equation 10. Parameter τ is a weight that controls the importance of the penalty term against the original cost function. We set $\tau = 0.5$ for this research, as a result of some preliminary testing. Note that the node asset-imbalance was raised to the power of $\gamma\ (>1)$ in order to apply higher penalties to highly unbalanced nodes. In this paper, we set $\gamma = 2$ for ease of computation and we note that this penalty function is slightly easier to compute than the one used in [24].

$$\psi_i = \sum_{j \in \mathcal{N}^+(i)} y_{ij} \quad - \sum_{j \in \mathcal{N}^-(i)} y_{ji} \tag{10}$$

4.2.2. Neighbourhood Definition

The neighbourhood which was used in the guided local search is deliberately the same as that used in [24], to allow a fair comparison to be made between the GLS and the tabu search method which was used in [24]. The neighbourhood is defined as the set of solutions which can be generated by either closing a currently open arc or opening a currently closed arc. Both closing or opening an arc could potentially result in an improved evaluation function value (Equation 9), either from a reduced fixed cost for the arc being closed or an improvement in any existing node imbalance from opening an arc.

Closing Arcs. To close an arc (i, j) that has a positive flow, the flow must be redirected to the remaining open arcs. The optimal flow re-distribution could be obtained by solving the model (5)-(7), however this would be prohibitively computationally expensive for a system which is designed to find solutions quickly. To alleviate the computational burden, as in the majority of previous approaches, a heuristic method is applied, based on a *residual network* [18] and the Dijkstra's shortest path algorithm, as follows: Let $resCap_{lt}$ denote the residual capacity of arc $(l,t) \in \mathscr{A}$ if arc (l,t) is open, or u_{lt} if arc (l,t) is closed. All commodities which have a positive flow on arc (i, j) are sorted into decreasing order of quantity and then handled in order, so that larger commodities are redirected first. For each of these commodities k, its entire flow d_k is removed from the network and a Γ^k residual network $\mathscr{G}^{\Gamma^k} = (\mathscr{N}, \mathscr{A}^{\Gamma^k})$ is constructed with the arcs in this residual network defined as follows:

$$\mathscr{A}^{\Gamma^k} = \{(l,t) \in \mathscr{A} \ \mid \ (l,t) \neq (i, j) \ \wedge \ resCap_{lt} \geq \Gamma^k\}.$$

where Γ^k is chosen such that $\Gamma^k = d_k$. The "cheapest" path on this residual network is then computed using Dijkstra shortest path algorithm and the entire flow of commodity k is redirected to this single path. If such a path cannot be found, the move is considered infeasible and the search goes back to the incumbent solution. The cost associated with each arc in this residual network is defined as follows:

$$c_{lt}^{\Gamma} = \begin{cases} f_{lt} + c_{lt} \cdot d^k & \text{if } (l,t) \in \mathscr{A}^{\Gamma^k} \text{ is closed,} \\ c_{lt} \cdot d^k & \text{if } (l,t) \in \mathscr{A}^{\Gamma^k} \text{ is open.} \end{cases} \tag{11}$$

The above flow redistribution is performed in turn for each commodity that has a positive flow on arc (i, j). If the procedure fails to redistribute the flow for any of them then the arc closing procedure is terminated and the move is considered infeasible.

Opening Arcs. Although opening an arc will increase the objective value due to the addition of the fixed cost for the added arc, it could potentially reduce the node imbalance penalty and hence lead to more feasible solutions. The optimal flow distribution probably changes when a closed arc is opened. However, re-solving the CMMCF model in order to determine the optimal flow distribution would be computationally prohibitive. Therefore, in this research the flow distribution is maintained when an arc is opened, with the only change being the addition of the incurred fixed cost of this arc and the potential for increased feasibility.

4.2.3. Algorithm overview

Starting from the current incumbent solution, the algorithm considers every neighbouring solution which can be generated using the opening and closing arc moves. The best solution in terms of GLS evaluation function (8) from this neighbourhood is then adopted as the new current solution, even if it is worse than the current incumbent solution. One of the problems which has to be faced by this kind of algorithm is that it is possible to cycle within a limited number of solutions within the search space, for instance a sequence of solutions such that each solution in the sequence is the best solution in the neighbourhood of the previous solution and the first and last solutions are identical. In particular, it is important to avoid two-solution cycles where each is the best solution in the neighbourhood of the other.

Since opening or closing a path involves opening or closing several arcs and in each case the flow is heuristically re-allocated (as in [24], which this work extends), the resulting solutions provide only approximations (although they are upper bounds) for the optimal cost for that network configuration. Before preceding with the next iteration, the CMMCF model is solved [1] for the adopted solution, in order to find the optimal flows. The ways in which this algorithm has been adapted and extended will be seen in more detail later.

5. Analysis of the GLS Extensions

In this section, we analyse and investigate various mechanisms for extending and improving the basic GLS algorithms, in isolation, in order to understand which of these elements contributes to the superior performance of the GLS against the multi-start tabu search method utilised in [24]. The mechanisms that we consider here are the aspiration criterion, memory length and tabu list for cycling prevention. We describe them one by one in the following subsections.

To be clear, the feasibility repair procedure, which is described later in this paper, is not applied at this stage, since the purpose of this section is primarily to study the behaviour of each mechanism. Since there is no random element to any of these GLS extensions, each instance only had to be executed once.

5.1. Aspiration criteria in GLS

One of the strengths of the guided local search is its ability to use a penalty function to take advantage of domain specific information as well as information gathered during the search. The penalty function punishes “unpopular” features present in a candidate solution. The popularity of a feature is determined by a combination of the pre-determined associated cost (utilising domain specific information) and the accumulated penalty values during the search, as explained in section 4.2.1.

Although this penalty function is, in general, effective in guiding the search to escape from locally optimal regions, conflicts between the objective function and the penalty function mean that it could miss some high quality solutions, since a good solution with respect to the original objective (1) is not necessarily so with respect to the evaluation function (8). We therefore

[1] In this implementation, CMMCF was solved by `LP_Solve`, a free open source linear programming solver based on the revised simplex method and written in the `C` programming language. `LP_Solve` can be downloaded from http://lpsolve.sourceforge.net/5.5/

incorporated an aspiration criterion into the basic GLS method so that the search will adopt a candidate solution if it improves the current best solution in terms of the evaluation function $g(\mathbf{x},\mathbf{y})$, even if its augmented objective (i.e. $E(s)$) is worse than that of other neighbouring solutions. Figure 3 plots the values of the solutions which were adopted at each iteration for the test instance C50 for both the basic GLS and the GLS with the aspiration criterion. It appears that, when given a longer execution time, the GLS with the aspiration criterion was able to perform better than the basic GLS on average at the middle and later stages of the search, while the GLS without aspiration criteria was able to find a better overall-best-solution. However, both versions of GLS found a best solution at an early stage in the search and failed to improve upon it later. This suggests that the penalty assignments used in the guided local search become ineffective after a certain number of iterations.

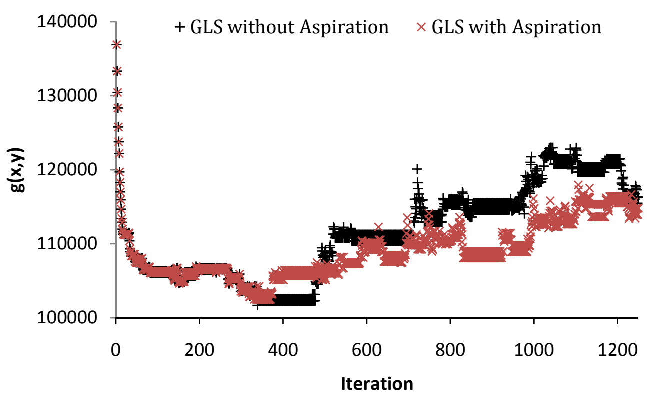


Figure 3: The behaviour of the basic GLS with and without aspiration criteria ($\alpha = 0.2$) for C50

Figure 4 shows a comparison of the GLS methods, with and without aspiration criteria, when executed with different α values. Firstly, it can be observed that the algorithm performed differently with different α values (the scaling parameter to set λ in function (8), see section 4.1). $\alpha = 0$ corresponds to a simple best-descent local search method, which performed far worse than the guided local search methods (i.e. when $\alpha > 0$). In fact it even failed to generate a feasible solution. Although we cannot draw a conclusion about which variant of GLS performs the best in terms of the function $g(\mathbf{x},\mathbf{y})$, it is clear that the addition of the aspiration criterion to the GLS helped to obtain feasible solutions. The GLS with the aspiration criterion was able to return a feasible solution for every α value tested while the basic GLS failed to obtain a feasible

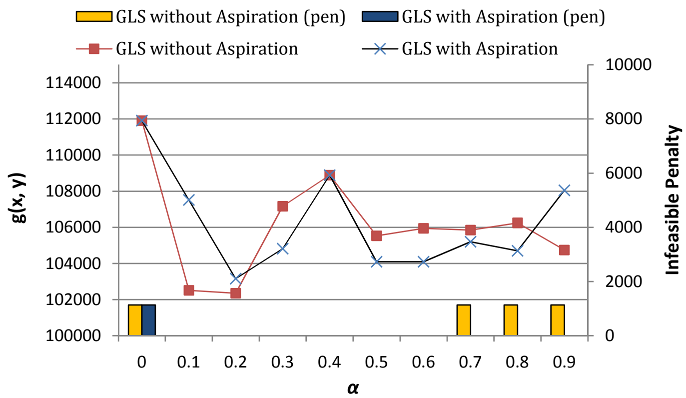


Figure 4: The GLS with and without aspiration criteria under different α values for C50.

solution on 3 occasions, when α was relatively large ($\alpha \geq 0.7$).

5.2. *Short-term memory*

In the basic GLS method in Figure 2, penalty values are the accumulated results from the entire search history. These penalty values get larger and larger, leading to undesirable dominance of the penalty term in the evaluation function $E(s)$ (Equation 8). This may explain why, in Figure 3, GLS failed to improve the best solution in the later stages of the search since the penalty term in the evaluation function (8) dominated the real objective function and can mislead the search away from regions with small objective values in terms of function $g(s)$ (Equation 8).

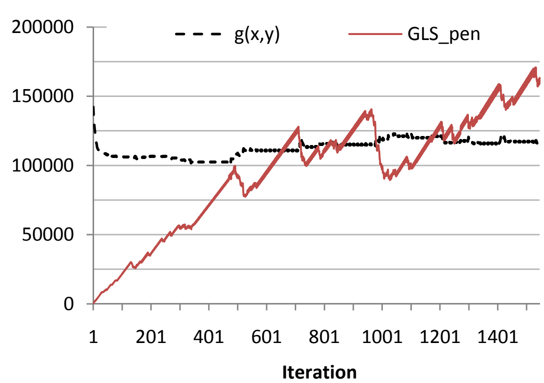


Figure 5: Evolution of the penalties in the basic GLS

Figure 5 depicts the evolution of two parts of the augmented evaluation function used by GLS (i.e. $g(\mathbf{x},\mathbf{y})$ and *GLS_pen* in Equation (8)). It can be observed that, after about 500 iterations, the penalty term has reached a value that is similar to the original evaluation function $g(\mathbf{x},\mathbf{y})$ and it first exceeds it around 670 iterations. At around 1300 iterations the penalty term has completely dominated the evaluation function, at which stage the search is basically misled to regions that contain solution features with low feature costs h_r. The original evaluation function then plays little role in guiding the search. This provides a reason why in Figure 3 the basic version of the GLS failed to improve upon the best solution during the middle and later stage of the search.

Mills et al. [23] observed success in handling the escalating penalty problem from utilising a GLS with a short-term memory schema, where the penalty value for a feature decreased when the feature had not been punished in the last ML iterations. We implemented a similar mechanism within our GLS method: A first-in-first-out queue of fixed capacity is used to record the list of features which have been penalised in the past ML iterations. When a feature is chosen to be penalised by the GLS (i.e. its penalty is increased by 1), it is pushed into the queue. If this would result in the queue exceeding its capacity then the oldest feature in the queue is removed and its penalty is decreased by 1.

Figure 6 depicts the algorithmic performance of the GLS with different memory length sizes and different scaling parameters α for four different instances of different sizes (C37, C45, C50, C60). It can be seen that the memory length does affect the performance in most cases, especially when $\alpha \leq 0.5$. Although the performance of the GLS was more erratic when the memory length (ML) was between 50 and 750, in general it seems that the optimal value for ML depends upon the scaling parameter α. With $\alpha > 0.5$, the GLS performance depended almost entirely upon the value of α rather than the ML.

In general, one may summarise that, firstly, when α is small, the optimal memory length tends to be larger; while when α is large, the optimal memory length is small. This may be explained from the fact that when α is large, a small memory length helps to prevent the penalty term from dominating the evaluation function $E(\mathbf{x},\mathbf{y})$. Similarly, when α is small, the GLS needs a longer memory length in order to build a penalty term that is large enough to impact on the search. Secondly, determining the most appropriate values for the ML parameter is difficult since it appears to be problem-instance dependent.

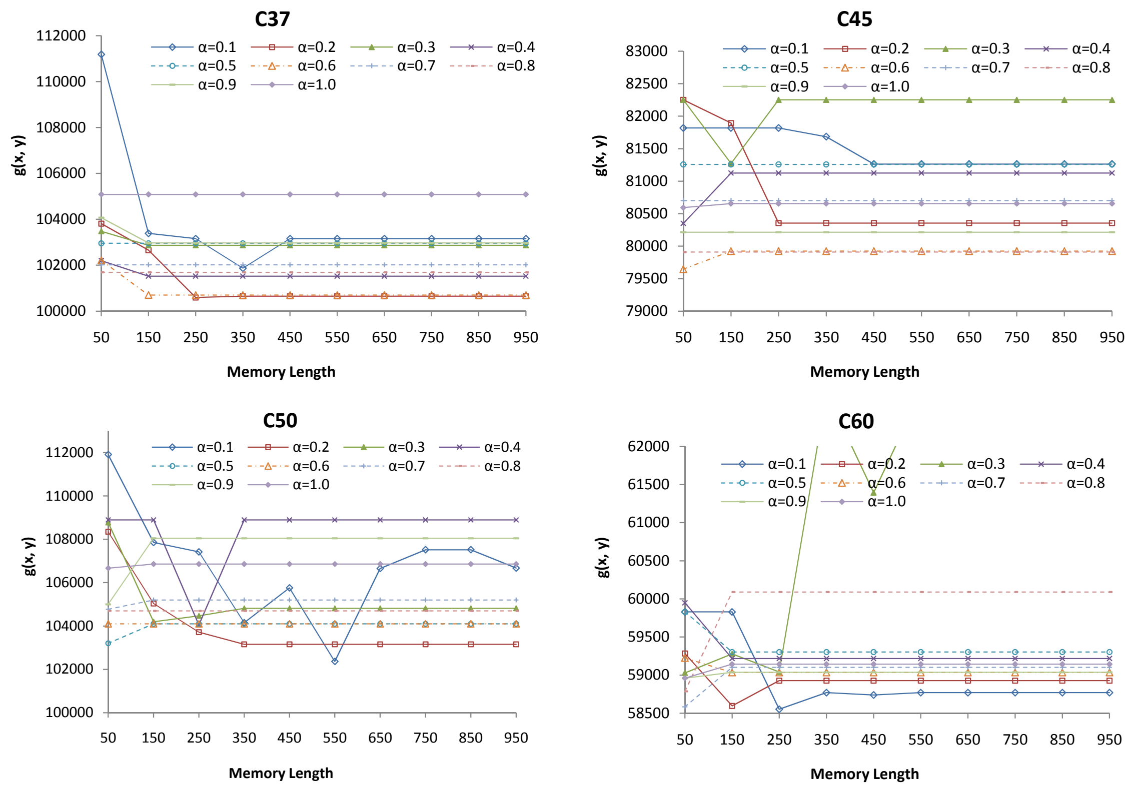


Figure 6: The relationship between the memory length ML and α

5.3. Recovering feasibility

Relaxing the asset-balance constraint greatly increases the freedom and flexibility for designing more effective neighborhoods. The designed algorithm converges to a feasible solution (i.e. all nodes are asset-balanced) for most instances, however, there are some instances for which the algorithm struggles to find a high quality feasible solution. A specialised heuristic was therefore required to repair the solution and recover feasibility. In this paper, we used the same heuristic which was used by Pederson et al. [24]. The main idea of this heuristic is to repeatedly reduce the asset-imbalance of the most unbalanced node (i.e. the node of the highest $|\psi|$). This is achieved by closing (or opening) a path between this node and another unbalanced node with opposite ψ sign. Since there may be many paths between these two nodes, the heuristic only evaluates the four shortest paths, computed by solving the shortest path problem in each of the following four different modified networks:

(1) *Small commodity flow network*. This network consists of all the open arcs of the current solution. The associated costs for each arc is set to the total flow on it. Therefore, closing

the shortest path in this network will require a relatively small amount of flow re-direction and hence lead to an increased chance for a successful flow-redistribution.

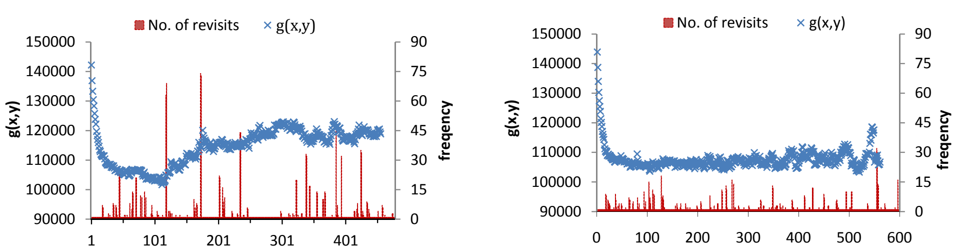
(2) *High fixed cost network.* This network is identical to (1) except that the associated arc costs are set to the fixed cost of the arc.

(3) *Small variable cost network.* This network is generated from the set of all closed arcs with the arc costs set to be the variable cost (or the average variable cost across all commodities). Opening the shortest path in this network could provide a cheaper commodity path (and hence lower variable costs) even though it will increase the fixed costs.

(4) *Low fixed cost network.* This network is similar to (3) except that the associated arc costs are set to the fixed cost of the arcs. The shortest path in this network represents the cheapest possible way (in terms of fixed costs) to open a new path between the two nodes.

5.4. Defeating the local optimal region/cycle trap

In order to analyse how efficiently the GLS escapes from locally optimal regions, we also carried out experiments based on some of the 24 C-set benchmark problems used in [24] [2]. All solutions which were adopted (i.e. which were the best solution in the neighbourhood of the previous solution and were chosen as the new current solution) during the search were recorded, together with the number of times that they were visited. All algorithms were evaluated on the same machine, with the same permitted computational time.



(a) Simple GLS (b) Multi-start GLS

Figure 7: The number of visits to each local optima by the simple GLS and M-GLS (C50)

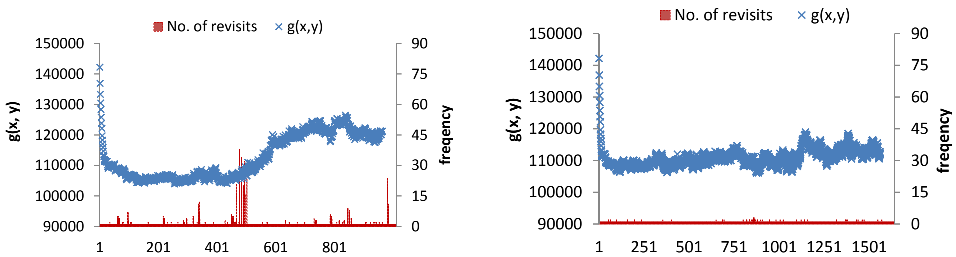
We initially tested a simple GLS approach and the multi-start GLS (denoted by M-GLS) proposed in [7]. The results are shown in Figures 7 (a) and (b), respectively. In each case,

[2]Representative results of only one instance (C50) are presented here due to space limitations. Similar performance was also observed for other instances.

the horizontal axis represents the list of solutions which were adopted during the search. One can see that both approaches re-adopted the same solution many times and some solutions were re-adopted over 80 times for the simple GLS. On average, the simple GLS adopted each of the solutions 2.3 times. By comparison, the average number of adoptions per solution for the M-GLS was even higher (2.9), but the high quality solutions tended to be adopted more often, indicating an improved ability to escape from lower quality regions. This may be one of the reasons for the improved performance over the simple GLS. However, overall, both versions of GLS wasted a significant amount of time evaluating the same solutions multiple times. In summary, the simple GLS seemed to converge to good local solutions very quickly but was not so efficient at escaping from poorer regions of the solution space.

5.5. Tabu assisted GLS (T-GLS)

Since even M-GLS could not effectively prevent the re-adoption of solutions, we borrowed the idea of a tabu list from the tabu search metaheuristic and introduced it into the simple GLS. The tabu list contains a list of arcs in the current solution which have been directly modified recently as a result of the 'opening an arc' or 'closing an arc' moves. Arcs which were opened due to flow redistribution (see section 4.2.2) are not included in the tabu list as this would be unnecessarily restrictive. The length of the tabu list is fixed to a predefined parameter, called *TabuLen* and is maintained on a first-in-first-out basis. To get an idea of the ability of the tabu list to avoid repetition of solutions, a number of different tabu lengths were considered for two sample instances (C37 and C50). Figures 8 (a) and (b) plot the objective values and the number of revisits by GLS with $TabuLen = 2$ and $TabuLen = 9$, respectively for C50. It can be seen that even a tabu list of length 2 was effective in reducing much of the re-adoption of solutions. When *TabuLen* was increased to 9, the majority of solutions were adopted only once. Experimental tests on instances C37 and C50 with tabu length values from 1 to 20 indicated that the behaviour with different tabu lengths seemed to be problem instance-dependent since there was no obvious overall best value for *TabuLen*. However, $TabuLen = 2$ performed well in general and was already very effective in preventing cycling and repetitions and obtained a better average objective function value than $TabuLen = 9$ in T_GLS over all instances (see table 2), so the final algorithm adopted $TabuLen = 2$. Since the multi-start mechanism in the final algorithm provides further diversification, a longer tabu list was considered unnecessary for this



(a) TabuLen=2 (b) TabuLen=9

Figure 8: The number of times each solution was adopted by the tabu assisted GLS (C50)

paper.

Table 2 provides more detailed results for the application of different GLS variants to the 24 C-set instances which are widely used in the service network design literature. Since there is no random element in either of the GLS variants, again only one run was carried out per instance. To compare the performance of different variants, we used three measures; the average objective function value $g(s)$, the number of infeasible solutions (i.e. the number of cases when the algorithm failed to obtain a feasible solution, labelled "# of infeas") and the average rank over all instances.

Note that the average objective value was calculated over instances for which feasible solutions were obtained by each of five GLS variants. The average rank was calculated as follows: firstly, the five variants were sorted for each instance according to the objective values obtained by each variant, the best variant being ranked 1 and the worst 5, with ties being assigned equal rank values. Infeasible solutions are considered as ties regardless of their objective value.

We can make the following observations from Table 2: 1) Although no single algorithm performed the best for every instance, the tabu assisted GLS with $TabuLen = 2$ outperformed the other 4 variants on average. Not only did it obtain more feasible solutions, but it also performed the best in terms of both the average objective values and the average ranking. The tabu assisted GLS with $TabuLen = 9$ performed better than GLS but appears to be too restrictive. In fact, additional experimental tests on instances C37 and C50 with other tabu length values showed that the algorithm generally performed better with $TabuLen = 2$. 2) The introduction of aspiration criteria to the GLS improved the performance slightly. 3) Contrary to [23], the short-term memory mechanism does not seem to have improved the performance of the GLS, at least for the parameters that were tested for this paper. This may be due to the fact that the same ML setting was used for all the instances and this parameter seems to be very sensitive to different problem instances. 4) All five variants failed to find feasible solutions for all of the instances, which indicated that more development was required. We address this in the next section.

Table 2: Computational details of different GLS variants for the 24 benchmark instances (C-Set). GLS stands for the basic GLS, GLS_AC is the GLS with Aspiration Criteria, SM_GLS is similar to GLS except that a short-term memory is used (ML=500), T_GLS is a tabu assisted GLS where both tabu lengths of 2 and 9 are tested.

Instance	**Feature**	**GLS**	**GLS_AC**	**SM_GLS** (ML=500)	**T_GLS** TabuLen=2	**T_GLS** TabuLen=9
C37	C20,230,200,V,L	**100649**	**100649**	**100649**	102138.0	102259.0
C38	C20,230,200,F,L	145872.0	145872.0	145872.0	**143325.0**	151886.0
C39	C20,230,200,V,T	104863.0	104863.0	104863.0	**104558.0**	104900.0
C40	C20,230,200,F,T	146884.0	146884.0	146884.0	**144725.0**	146469.0
C45	C20,300,200,V,L	**80356.0**	**80356.0**	**80356.0**	80760.0	81004.0
C46	C20,300,200,F,L	127356.0	127356.0	127356.0	**124764.0**	126318.0
C47	C20,300,200,V,T	79699.5	79699.5	79699.5	**78607.5**	79858.3
C48	C20,300,200,F,T	131878.0	131878.0	131878.0	120178.0	**118794.0**
C49	C30,520,100,V,L	56166.0	56166.0	56166.0	56168.0	**56127.0**
C50	C30,520,100,F,L	**102354.0**	**102354.0**	**102354.0**	103828.0	106542.5
C51	C30,520,100,V,T	inf.	inf.	inf.	inf.	53940.0
C52	C30,520,100,F,T	108223.0	109232.0	108223.0	**106943.0**	108870.0
C53	C30,520,400,V,L	120827.8	120827.8	120827.8	**119081.2**	120269.2
C54	C30,520,400,F,L	162213.0	161199.0	162213.0	162213.0	**160908.8**
C55	C30,520,400,V,T	inf.	inf.	inf.	inf.	inf.
C56	C30,520,400,F,T	**166721.3**	inf.	inf.	**166721.3**	**166721.3**
C57	C30,700,100,V,L	49327.0	49327.0	49327.0	49459.0	**49236.0**
C58	C30,700,100,F,L	65270.0	**63660.0**	65270.0	64658.0	64367.0
C59	C30,700,100,V,T	inf.	48365.0	inf.	48125.0	**47857.0**
C60	C30,700,100,F,T	58927.0	**58188.0**	58927.0	58603.0	58428.0
C61	C30,700,400,V,L	103317.0	104311.0	103331.5	**103030.0**	104190.0
C62	C30,700,400,F,L	153204.0	151731.5	153204.0	**151147.0**	153194.0
C63	C30,700,400,V,T	inf.	inf.	inf.	inf.	inf.
C64	C30,700,400,F,T	**143447.0**	145358.0	144102.0	**143447.0**	inf.
	average obj*	105410.4	105253.0	105411.2	**104121.4**	105201.2
	# of infeas	4	4	5	**3**	**3**
	average rank	3.3	2.9	3.4	**2.4**	3.1

inf.: indicates an infeasible solution.

*: instances for which at least one algorithm fails to solve are not averaged.

5.6. Putting Everything Together

Table 2 shows that, although adding an aspiration criterion and a tabu list to the basic GLS improved the performance, the algorithms still failed to find a feasible solution for some prob-

begin
  Generate an initial solution s_0 by rounding design
  variables of the LP solution of model (1)-(6);
  Set $s' := s_0$; $s^* = s_0$;
  while TimeAvailable **do**
    `/* Guided Local Search Phase */`
    while TimeAvailable && *nonImpCount* $< K$ **do**
      Set $s' := s$;
      foreach arc $(i, j) \in \mathscr{A}$ **do** $\{p_{ij} := 0; h_{ij} := f_{ij};\}$
      $s \leftarrow BestNeighbourTabu(s, E(s))$;
      $s \leftarrow CMMCF(s)$;
      if $g(s) < g(s')$ set $s' := s$;
      Update the best feasible solution s^* with respect
      to $z(X,Y)$;
      `/* Update GLS Parameters */`
      foreach arc in $\mathscr{A}$ **find** the arc (i, j) that maximises
        $util_{ij} = I_{ij}(s') \cdot h_{ij}/(1 + p_{ij})$;
      Set $p_{ij} := p_{ij} + 1$; Set $nonImpCount := nonImpCount + 1$
    end while
    `/* Feasibility Recovery Phase */`
    $s \leftarrow FeasibilityRecovery(s')$
    Update the best feasible solution s^*;
    $s := Perturbation(s')$;
  end while
end
return s^*.

Figure 9: A tabu assisted multi-start guided local search algorithm for SNDP.

lem instances. GLS_AC failed to return a feasible solution for 4 instances (out of 24) and T_GLS failed on three occasions. Since finding a feasible solution is probably more important in practice than improving the objective value, specific feasibility recovery procedures may be required. In this research, we propose the adoption of the following multi-start tabu assisted guided local search method. The algorithm is characterized by 1) a multi-start framework with each iteration consisting of three coordinated phases: a local search phase, a feasibility recovering phase and a perturbation procedure. 2) a short tabu list hybridised with GLS in order to prevent cycling. The algorithm is described in detail in the next section.

5.6.1. *Tabu assisted multi-start guided local search (TA_MGLS)*

The proposed algorithm has three phases (see Figure 9). The first phase, the guided local search phase, is just one of the GLS variants in Table 2 (T_GLS, which provides overall best results). The second phase is a feasibility recovery phase, which recovers the feasibility as a priority and minimises the objective value as a secondary task (see section 5.3). This is followed by a perturbation procedure in the third phase. These phases are then put into a multi-start framework. The initial solution is built by solving the relaxed LP model (i.e. the integrity constraint of design variables is relaxed) and then rounding the values of the design variables to binary integers. The three phases described above are then executed repeatedly until the computational time is exhausted. The guided local search phase stops when either the computational time limit has been reached or the number of consecutive non-improving iterations reaches a given value L.

In the guided local search, the neighborhood is defined by closing/opening arcs as described in section 4.2.2. Procedure $BestNeighbourTabu(s, E(s))$ returns the best non-tabu neighbour of the current solution s according to the augmented objective function $E(s)$. Since the flow distribution is only estimated during neighbourhood exploration, the solution returned by $BestNeighbourTabu(s, E(s))$ is then re-optimised by solving the corresponding CMMCF model. During the guided local search phase, the best solution with regard to $g(s)$ and the best feasible solution with regard to $z(\mathbf{x}, \mathbf{y})$ are recorded. Procedure $FeasibilityRecovery(s)$ is then applied after the guided local search phase to recover the solution feasibility (i.e. the asset-balance) using the heuristic described in section 5.3. If this phase finds a solution that is better than the best feasible solution, in terms of the original objective $z(\mathbf{x}, \mathbf{y})$, then the best feasible solution is updated.

After the $FeasibilityRecovery(s)$ finishes, the solution is perturbed using the path-opening heuristic (see section 5.6.2) to prepare for the next round of the guided local search. The algorithm stops when the computational time has been exhausted.

5.6.2. *Opening a path*

For two nodes i, j with opposite asset imbalances (i.e. $\psi_i \cdot \psi_j < 0$), opening or closing a path between i and j will reduce the asset imbalance penalty for both nodes (note that the direction of the path will depend upon the actual signs of ψ_i and ψ_j). If, however, there is no direct arc between the nodes, a neighbourhood move of closing/opening a single arc will not reduce the node asset unbalance, so it is sometimes beneficial to open a path between two nodes. In addition, sometimes when the local search stagnates in a locally optimal region, it is useful to randomise the incumbent solution and restart the search to escape from these regions. Therefore, we also introduced another neighbourhood operator, which selects a random commodity and opens one of its top 10 shortest paths with equal probability. The corresponding CMMCF model is then solved to obtain the optimal flow distribution. Since this operator is only called occasionally, solving the CMMCF problem in this procedure will have a much lower impact on the computational expense of the algorithm. The top 10 shortest paths for a given commodity are computed independently without the consideration of other commodities and are, therefore, only computed once, at the beginning of the search.

Table 3: A comparison of TA_MGLS against Xpress MIP solver, Tabu Search (TS) and Multi-start GLS (M_GLS). Results by Xpress MIP and Tabu Search were drawn from [24]. Results by M_GLS were drawn from [7].

Inst.	**Feature**	**Xpress MIP**	**TS (1 run)**	**M_GLS (10 runs)**		**TA_MGLS (10 runs)**	
				best	**average**	**best**	**average**
C37	C20,230,200,V,L	101112	102919	100771	102311	**98798**	**100774**
C38	C20,230,200,F,L	153534	150764	143017	145763	**142216**	**144699**
C39	C20,230,200,V,T	105840	**103371**	103428	104030	**103063**	104198
C40	C20,230,200,F,T	154026	149942	143446	145401	**141853**	**143690**
C45	C20,300,200,V,L	81184	82533	79020	80764	**78787**	**79692**
C46	C20,300,200,F,L	131876	128757	125290	127791	**124580**	**125362**
C47	C20,300,200,V,T	78675	**78571**	77839	78906	**77209**	78837
C48	C20,300,200,F,T	127412	**116338**	116712	119092	**114601**	117156
C49	C30,520,100,V,L	**55138**	55981	55437	55806	55422	55869
C50	C30,520,100,F,L	n/a	104533	**99821**	102732	100342	**102497**
C51	C30,520,100,V,T	**53125**	54493	53644	54073	53744	54027
C52	C30,520,100,F,T	106761	**105167**	104753	105924	**103996**	105451
C53	C30,520,400,V,L	n/a	119735	119344	121214	**117562**	**118551**
C54	C30,520,400,F,L	n/a	**162360**	161731	162373	**160339**	162480
C55	C30,520,400,V,T	n/a	**120421**	122877	593677*	4366710*	814149*
C56	C30,520,400,F,T	n/a	**161978**	165894	552913*	165757	1002886*
C57	C30,700,100,V,L	**48849**	49429	49451	49575	49221	49269
C58	C30,700,100,F,L	65516	63889	63516	64102	**62205**	**63143**
C59	C30,700,100,V,T	**47052**	48202	47518	47833	47518	47859
C60	C30,700,100,F,T	**57447**	58204	58017	58563	57673	57938
C61	C30,700,400,V,L	n/a	103932	**103136**	**103577**	103352	103621
C62	C30,700,400,F,L	n/a	157043	150449	596588*	**148809**	**150867**
C63	C30,700,400,V,T	n/a	**103085**	103581	537477*	103323	784872*
C64	C30,700,400,F,T	n/a	**141917**	142575	144135	143717	144368

* indicates an infeasible solution

5.6.3. *Results Analysis*

Table 3 provides a comparison between the hybrid tabu assisted guided local search, the tabu search [24] and the guided local search [7]. Both Xpress MIP and TS were run on a Pentium IV 2.26GHz PC with 3600 seconds CPU time [24]. Both M_GLS and TA_MGLS are single-threaded programs and were run on a PC with a 1.8GHz Intel Core 2 CPU and a 2400-second time limit. Both algorithms were run 10 times with different random seeds. Due to space limitations, only the best and average results are included in Table 3. More detailed results can be found in Table 5.

Firstly, compared to the other three algorithms, TA_MGLS had the best average performance for 9 instances (out of 24). The best results (out of 10 runs) by TA_MGLS beat the other three algorithms on 13 instances. On average, TA_MGLS outperformed M_GLS on 15 instances but was inferior to M_GLS on 6 instances. For 3 instances, both algorithms returned infeasible solutions on some runs. This demonstrated the effectiveness of the added features in TA_MGLS.

Secondly, it can also be seen that the cost of the features chosen in GLS plays a role in its performance. TA_MGLS seems to solve the fixed-cost dominated instances better than variable-cost dominated instances. Out of the 9 instances on which TA_MGLS performed better, 6 instances were fixed-cost dominated instances. This may imply that better performance could be achieved by choosing feature costs based upon the problem characteristics.

Nevertheless, we notice that TA_MGLS sometimes failed to find feasible solutions for 3 of the instances, and consistently failed on one instance. This is problematic for real-world applications, so more development and investigation was required. After more detailed analysis, we found that this was caused by the poor performance of LP_Solve. We now give the results and analyses in the next section.

6. Computational Time Distribution

We monitored the time that was spent by LP_Solve and GLS respectively. It turned out that LP_Solve struggled to solve some of the CMMCF problem instances (e.g. C37, C53, C54, C55, C56, C63, and C64), spending more than 85% of the total time allowed, leaving very little time for the GLS and the feasibility recovery heuristic to search for high quality feasible solutions. To further confirm the conjecture that this was the problem, we replaced LP_Solve with CPLEX12 as the CMMCF solver and kept the remaining configuration the same. To ensure a fair comparison, the CPLEX thread count was set to 1 to prevent it using more than one CPU core. Table 4 shows a comparison of the two variants for the benchmark instance set. It can be seen that not only was the proposed algorithm able to find a feasible solution for every instance in every run, but in addition, both the average and best results were considerably improved.

Table 4: A comparison of LP_Solve and CPLEX as LP Solvers

Inst.	**TS**	**TA_MGLS with LP_Solve**			**TA_MGLS with CPLEX**		
	(1 run)	best	avg	Lptime	best	avg	LPtime
C37	102919	98797.8	100774	85%	**98760.0**	**99622.2**	31%
C38	150764	142216	144699	81%	**142113.0**	143867.0	30%
C39	103371	103063	104198	86%	**102137.3**	**102833.1**	31%
C40	149942	141853	**143690**	87%	**141802.0**	143839.5	37%
C45	82533	**78787**	**79692**	86%	79029.5	79895.4	34%
C46	128757	124580	125362	84%	**121773.0**	**124454.3**	34%
C47	**78571**	77209.3	78837	85%	**77066.0**	**78302.5**	30%
C48	116338	114601	117156	86%	**114465.0**	**115836.2**	31%
C49	55981	**55422**	**55869**	48%	55732.0	55985.6	9%
C50	104533	100342	102497	58%	**100290.0**	**102016.9**	12%
C51	54493	**53744**	**54027.2**	58%	54372.0	54707.7	11%
C52	**105167**	**103996**	105451	61%	104574.0	105422.8	11%
C53	119735	117561.71	118551	87%	**116196.0**	**116915.4**	47%
C54	162360	160339	162480	83%	**154941.0**	**156008.5**	37%
C55	120421	436679.8*	814149.1*	86%	**118335.7**	**118894.1**	42%
C56	161978	165757.3	1002886.0*	80%	**157939.6**	**159426.5**	47%
C57	49429	**49221**	**49269**	33%	49385.0	49456.9	6%
C58	63889	62205	63143	37%	**62055.0**	**62773.6**	7%
C59	48202	**47518**	47859	49%	47519.0	**47727.8**	7%
C60	58204	57673	**57938.0**	46%	**57571.0**	58046.0	7%
C61	103932	103352.2	103621	75%	**101609.5**	**102215.9**	21%
C62	157043	148809.3	150867	78%	**142563.2**	**144754.7**	36%
C63	103085	103323	784871.7*	90%	**98656.8**	**99726.1**	28%
C64	141917	143717	144368	78%	**135777.5**	**136727.1**	29%

* indicates an infeasible solution being returned by the algorithm.

Table 5: More detailed results by M_GLS and TA_MGLS (with LP_Solve and CPLEX). Results are based on 10 independent runs.

	M_GLS				**TA_MGLS with LP_Solve**				**TA_MGLS with CPLEX**			
	best	avg	worst	stdev	best	avg	worst	stdev	best	avg	worst	stdev
C37	100771.0	102311.3	105254.0	1436.7	98797.8	100773.8	102076.0	1092.6	**98760.0**	**99622.2**	101605.5	865.1
C38	143017.0	145762.7	152056.0	2546.7	142216.0	144698.9	146112.0	1186.0	**142113.0**	**143867.0**	146822.7	1592.8
C39	103428.0	104030.1	104695.0	418.4	103063.0	104198.3	104662.0	610.3	**102137.3**	**102833.1**	104424.0	659.2
C40	143446.0	145400.8	147081.3	1226.3	141853.0	**143690.1**	146348.0	1361.6	**141802.0**	143839.5	146141.0	1161.7
C45	79020.0	80764.1	81681.0	830.1	**78787.0**	**79691.8**	81052.5	653.5	79029.5	79895.4	80888.0	692.3
C46	125290.0	127790.6	130779.5	1890.6	124580.0	125361.7	126204.5	490.7	**121773.0**	**124454.3**	127607.0	1768.6
C47	77839.0	78905.7	80006.0	600.5	77209.3	78837.4	79761.0	703.3	**77066.0**	**78302.5**	80009.0	1124.9
C48	116712.0	119091.7	122125.7	1938.5	114601.0	117155.9	120070.0	1644.4	**114465.0**	**115836.2**	117046.0	883.8
C49	55437.0	**55806.2**	55937.0	140.8	**55422.0**	55869.1	56010.0	183.6	55732.0	55985.6	56260.0	180.3
C50	**99820.6**	102731.9	104435.0	1503.8	100342.0	102497.5	104495.0	1230.6	100290.0	**102016.9**	102838.0	769.1
C51	**53644.0**	54073.3	54491.0	225.4	53744.0	**54027.2**	54157.0	111.5	54372.0	54707.7	54838.0	179.3
C52	104753.0	105923.6	107276.0	861.2	**103996.0**	105450.9	106999.0	914.9	104574.0	**105422.8**	106477.0	643.6
C53	119344.0	121214.1	121965.2	1089.0	117561.7	118551.0	120502.0	1314.6	**116196.0**	**116915.4**	117888.0	566.0
C54	161730.8	162372.7	163138.5	427.6	160339.0	162479.7	166397.0	2748.4	**154941.0**	**156008.5**	157630.0	820.4
C55	122877.0	593676.6*	750987.0*	266496.5	436679.8*	814149.1*	1066683.4	289057.5	**118335.7**	**118894.1**	120445.3	648.2
C56	165894.3	552913.3*	1454985.1*	622488.8	165757.3	1002886.0*	1454313.2*	610968.3	**157939.6**	**159426.5**	161271.8	951.8
C57	49451.0	49575.4	49930.0	172.6	**49221.0**	**49268.7**	49274.0	16.8	49385.0	49456.9	49482.0	38.9
C58	63515.5	64101.7	64455.0	298.0	62205.0	63143.0	63775.0	495.1	**62055.0**	**62773.6**	63397.0	424.6
C59	**47518.0**	47833.2	48006.0	164.2	**47518.0**	47859.0	47965.0	154.6	47519.0	**47727.8**	47937.0	137.6
C60	58017.0	58562.5	59216.0	377.1	57673.0	**57938.0**	58120.0	138.7	**57571.0**	58046.0	58447.0	234.6
C61	103136.0	103576.8	105493.3	737.6	103352.2	103621.1	104486.0	334.6	**101609.5**	**102215.9**	103007.7	450.3
C62	150449.0	596588.2*	2387564.0*	937076.6	148809.3	150867.2	151425.0	814.1	**142563.2**	**144754.7**	147828.3	1574.0
C63	103581.0	537476.8*	723741.0*	216658.4	103323.0	784871.7*	1342439.8*	433081.3	**98656.8**	**99726.1**	100590.4	652.2
C64	142575.0	144134.6	145275.0	788.2	143717.0	144367.9	144519.7	245.6	**135777.5**	**136727.1**	138003.5	670.5

* indicates an infeasible solution being returned by the algorithm.

7. Discussions and Future Work

In this research, we carried out experiments to look at several components and aspects that could potentially speedup the GLS approach for service network design problems. These include a tabu list, short-term memory and aspiration criteria. In particular, we found that in both the simple GLS and its multi-start version, time was wasted due to the repeated re-adoption of the same solutions. We proposed a tabu assisted GLS schema within a multi-start framework to prevent this problem. Results have demonstrated the superiority of the method.

In addition, our observations showed that `LP_Solve` struggled on some problem instances, for which the majority of computational time (more than 85%) was used when solving CMMCF problems. Our tests with CPLEX12 showed that a faster LP solver was able to further improve the performance of our proposed method.

In future, we will investigate more neighbourhood structures in the GLS approach. In particular, we will consider multiple ways for flow redirection on opening/closing an arc. In addition, if certain criteria are met, more accurate neighbourhood evaluations may be used by applying CPLEX to solve the CMMCF problem. Another potential research direction is adopting stochastic programming to address some of uncertainties in SNDP.

References

[1] R. Ahuja, T. Magnanti, J. Orlin, Network Flows: Theory, Algorithms and Applications, Prentice Hall, 1993.

[2] J. Andersen, M. Christiansen, T. G. Crainic, R. Gronhaug, Branch and price for service network design with asset management constraints, Transportation Science 45 (1) (2011) 33–49.

[3] J. Andersen, T. G. Crainic, M. Christiansen, Service network design with management and coordination of multiple fleets, European Journal of Operational Research 193 (2) (2009) 377 – 389.

[4] A. P. Armacost, C. Barnhart, K. A. Ware, Composite Variable Formulations for Express Shipment Service Network Design, Transportation Science 36 (1) (2002) 1–20.

[5] A. P. Armacost, C. Barnhart, K. A. Ware, A. M. Wilson, UPS Optimizes Its Air Network., Interfaces 34 (1) (2004) 15–25.

[6] R. Bai, G. Kendall, Tabu Assisted Guided Local Search Approaches for Freight Service Network Design, in: n the CD proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling, 10th-13th August, 2010, Belfast, Northern Ireland, Belfast, Northern Ireland, 468–471, 2010.

[7] R. Bai, G. Kendall, J. Li, A Guided Local Search Approach for Service Network Design Problem with Asset Balancing, in: 2010 International Conference on Logistics Systems and Intelligent Management (ICLSIM 2010), January 9-10, Harbin, China, 110–115, 2010.

[8] L. Barcos, V. Rodriguez, M. Alvarez, F. Robuste, Routing design for less-than-truckload motor carriers using ant colony optimization, Transportation Research Part E 46 (2010) 367–383.

[9] C. Barnhart, N. Krishnan, D. Kim, K. Ware, Network Design for Express Shipment Delivery, Computational Optimization and Application 21 (2002) 239–262.

[10] S. Bock, Real-time control of freight frowarder transportation networks by integrating multimodal transport chains, European Journal of Operational Research 200 (2010) 733–746.

[11] S. Chiou, A bi-level programming for logistics network design with system-optimized flows, Information Sciences 179 (2009) 2434–2441.

[12] T. G. Crainic, Service network design in freight transportation, European Journal of Operational Research 122 (2) (2000) 272 – 288.

[13] T. G. Crainic, M. Gendreau, J. M. Farvolden, A Simplex-Based Tabu Search Method for Capacitated Network Design, INFORMS Journal on Computing 12 (3) (2000) 223–236.

[14] T. G. Crainic, M. Gendreau, P. Soriano, M. Toulouse, A tabu search procedure for multicommodity location/allocation with balancing requirements, Ann. Oper. Res. 41 (1-4) (1993) 359–383.

[15] T. G. Crainic, K. H. Kim, Chapter 8 Intermodal Transportation, in: C. Barnhart, G. Laporte (Eds.), Transportation, vol. 14 of *Handbooks in Operations Research and Management Science*, Elsevier, 467 – 537, 2007.

[16] T. G. Crainic, J.-M. Rousseau, Multicommodity, multimode freight transportation: A general modeling and algorithmic framework for the service network design problem, Transportation Research Part B: Methodological 20 (3) (1986) 225 – 242.

[17] T. G. Crainic, J. Roy, OR tools for tactical freight transportation planning, European Journal of Operational Research 33 (3) (1988) 290 – 297.

[18] I. Ghamlouche, T. G. Crainic, M. Gendreau, Cycle-based Neighbourhoods for Fixed-Charge Capacitated Multicommodity Network Design, Operations Research 51 (4) (2003) 655–667.

[19] I. Ghamlouche, T. G. Crainic, M. Gendreau, Path Relinking, Cycle-Based Neighbourhoods and Capacitated Multicommodity Network Design, Annals of Operations Research 131 (2004) 109–133.

[20] B. Jansen, P. C. J. Swinkels, G. J. A. Teeuwen, B. van Antwerpen de Fluiter, H. A. Fleuren, Operational planning of a large-scale multi-modal transportation system, European Journal of Operational Research 156 (1) (2004) 41–53, eURO Excellence in Practice Award 2001.

[21] D. Kim, C. Barnhart, K. Ware, G. Reinhardt, Multimodal Express Package Delivery: A Service Network Design Application, Transportation Science 33 (4) (1999) 391–407.

[22] C. Liu, Y. Fan, F. Ordonez, A two-stage stochastic programming model for transportation network protection, Computers & Operations Research 36 (5) (2009) 1582–1590, selected papers presented at the Tenth International Symposium on Locational Decisions (ISOLDE X).

[23] P. Mills, E. P. Tsang, J. Ford, Applying an Extended Guided Local Search to the Quadratic Assignment Problem, Annals of Operations Research 118 (2003) 121–135.

[24] M. B. Pedersen, T. G. Crainic, O. B. Madsen, Models and Tabu Search Metaheuristics for Service Network Design with Asset-Balance Requirements, Transportation Science 43 (4) (2009) 432–454.

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

[26] C. Voudouris, E. P. Tsang, Guided Local Search, in: F. Glover, G. Kochenberger (Eds.), Handbook of Metaheuristics, Kluwer, 185–218, 2003.

[27] N. Wieberneit, Service network design for freight transportation: a review, OR Spectrum 30 (2008) 77–112.